TECHNISCHE UNIVERSITÄT ILMENAU

Fakultät für Informatik und Automatisierung

Fachgebiet Rechnerarchitektur

# Bachelorarbeit

## Entwicklung und Realisierung einer FPGA-Anwendung für ein LVDS-Kommunikationssystem mit DSP-Ankopplung

vorgelegt von

Erik Wolf

Matrikel 40864

Betreuer:

Prof. Dr.-Ing. habil Wolfgang Fengler

Dr.-Ing. Bernd Däne

Ilmenau, den 10 Mai 2010

URN: urn:nbn:de:gbv:ilm1-2010200193

# Eidesstattliche Erklärung

Hiermit versichere ich, Erik Wolf, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als den angegebenen Hilfsmitteln angefertigt habe. Die aus anderen Quellen indirekt oder direkt übernommenen Vorlagen, Daten und Konzepte sind unter Angabe der jeweiligen Quelle gekennzeichnet.

Ilmenau, den 10. Mai 2010 . . . . . .

Erik Wolf

# Kurzfassung

Seit 2002 wird fachgebietsübergreifend im Sonderforschungsbereich 622 „Nanopositionier- und Nanomessmaschinen“ an der Technischen Universität Ilmenau gearbeitet. Ziel des von der Deutschen Forschungsgemeinschaft geförderten Bereichs ist die Erforschung und Erarbeitung von wissenschaftlich-technischen Grundlagen für deren Entwurf und die Realisierung.
Dieses hochgenaue Positioniersystem soll durch ein Mehrprozessorsystem gesteuert werden. Aufgabe des Fachgebietes Rechnerarchitektur ist es, unterschiedliche Alternativen für die Realisierung zu finden und zu bewerten. Dabei stehen neben der Implementierung echtzeitfähiger Betriebssystem- und Applikationssoftware auch die Entwicklung von leistungsfähiger Hardware und Kommunikationsprotokollen im Vordergrund.
Ziel dieser Arbeit ist es, das LVDS-basierte Kommunikationssystem eines Multi-DSP-Prototypen zu erweitern. Durch eine Brücke soll die Vermittlung zwischen DSP und Kommunikationsknoten bereitgestellt werden. Diese Brücke ermöglicht es dem DSP, Nachrichtenpakete im Kommunikationsring zu verschicken und zu empfangen. Zusätzlich soll der DSP mittels Interrupt über bestimmte Systemereignisse informiert werden.
Die Entwicklung der Vermittlungsstelle wird modellbasiert durchgeführt. Durch das fortlaufende Überprüfen des Verhaltens während der Entwurfsphase können Fehler frühzeitig erkannt und beseitigt werden. Die realisierte Funktionalität wird anschließend unter Verwendung der Hardwarebeschreibungssprache VHDL auf die Logikbausteine der Kommunikationsknoten synthetisiert.
Ein weiterer Aspekt dieser Arbeit ist die umfangreiche Simulation des bereits entwickelten Kommunikationsprotokolls. Durch entsprechende Testszenarien soll die korrekte Funktionalität der Kommunikationsknoten nachgewiesen werden.

# Abstract

Since 2002, the Technical University of Ilmenau interdisciplinary researchs in the special research field no. 622 "Nano Positioning and Nano Measurement Machines". The intention of the project, sponsored by the DFG (Deutsche Forschungsgemeinschaft), is to investigate the scientific and technical fundamentals of the design and the construction of those machines.
This highly-precisioned positioning-device is supposed to be controlled by a multiprocessing system. The task of the Departement of Computer Architecture is to find and evaluate different alternatives for the implementation. Besides the implementation of real-time capable operating systems and application software, the development of efficient hardware and communications-protocols is imperative important.
The aim of this thesis is to extend the LVDS-based communications-system of an multi-DSP-prototype. The transmission between DSP and communications-nodes is controlled via a bridge, which enables the DSP the to send messages into the communications-ring and to receive messages from other DSPs. Additionally the DSP will be informed about special system-events via an interrupt.
The development of the interposition is model-based, which provides the advantage to verify the behaviour and resolve the malfunctions already during the design-process. Finally the realized functionality is synthesized on the logic modules of the communications-nodes by using the hardware description language VHDL.
Another aspect of this thesis is the comprehensive simulation of an already available communications-protocol. With the help of several test scenarios the correct functionality of the communications-nodes is to be proved.

# Inhaltsverzeichnis

# 1. Einleitung

## 1.1. Überblick

Nanomess- und -positioniermaschinen kommen seit einigen Jahren eine wachsende Aufmerksamkeit in Wirtschaft und Forschung zuteil. Das Projekt der TU Ilmenau ist im Sonderforschungsbereich (SFB) 622 der Deutschen Forschungsgemeinschaft angesiedelt [TU 10]. An der erfolgreichen Entwicklung einer hochgenauen Nanopositioniermaschine arbeiten 40 Wissenschaftler aus 14 Fachgebieten interdisziplinär zusammen. Durch den hohen Grad an Präzision werden immer höhere Ansprüche an die Architekturen der Signal- und Informationsverarbeitung gestellt. Das Teilprojekt C1 am Fachgebiet für Rechnerarchitektur beschäftigt sich aus diesem Grund mit leistungsfähigen Mehrprozessorarchitekturen basierend auf digitalen Signalprozessoren.
Ziel dieser Arbeit ist es, das LVDS-basierte Kommunikationssystem eines Mehrprozessor-Prototypen zu erweitern. Im Gegensatz zu dem Vorgänger-Prototypen wird ein serieller Kommunikationsweg anstatt eines parallelen Bussystems verwendet. Dies bringt Vorteile in der Skalierbarkeit der Architektur mit sich.
Die Kernelemente sind Digitale Signalprozessoren (DSP) und Programmierbare Logikbausteine (FPGA). Für die Kommunikation zwischen mehreren Prozessoren liegen drei unabhängige Kommunikationssysteme vor. Diese Systeme sollen im Rahmen studentischer Arbeiten schrittweise implementiert werden. Durch die unterschiedlichen Topologien und Komplexitäten der Kommunkationswege kommen verschiedene Techniken für den Datenaustausch und das Übertragungsprotokoll zum Einsatz.

## 1.2. Aufgabenstellung

Ziel dieser Arbeit ist es, die in einer Vorarbeit entstandene LVDS-Kommunikationsschnittstelle für die Vernetzung der FPGAs zu erweitern. Jeder FPGA soll mittels der EMIF-Schnittstelle an das zugehörige DSP-Modul gekoppelt werden. Für die Entwicklung und Realisierung der EMIF-Bridge ist eine Einarbeitung in die Vorarbeiten von Marcus Müller [Mül09] (erste grundlegende Realisierung der EMIF-Bridge) und Raik Schulze [Sch10] (Entwicklung des Kommunikationsprotokolls für den LVDS-Ring) nötig. Zusätzlich erfolgt die Auseinandersetzung mit den verwendeten Tools und Aspekten der modellbasierten Entwurfsmethodik. Der bereits entworfene LVDS-Kommunikationsknoten soll mit der EMIF-Bridge in das Gesamtsystem integriert werden. Abschließend sollen Simulationen mit Matlab und Tests an der realen Hardware die korrekte Arbeitsweise des Kommunikationssystems sicherstellen.

# 2. Grundlagen

## 2.1. Hardware

Die Realisierung des Multi-DSP-Systems liegt zur Zeit in einem Hardware-Prototypen vor, der aus drei modular verbundenen Teilplatinen besteht:

- DSP-Modul (D.Module.6713)
- Basisboard
- Backplane

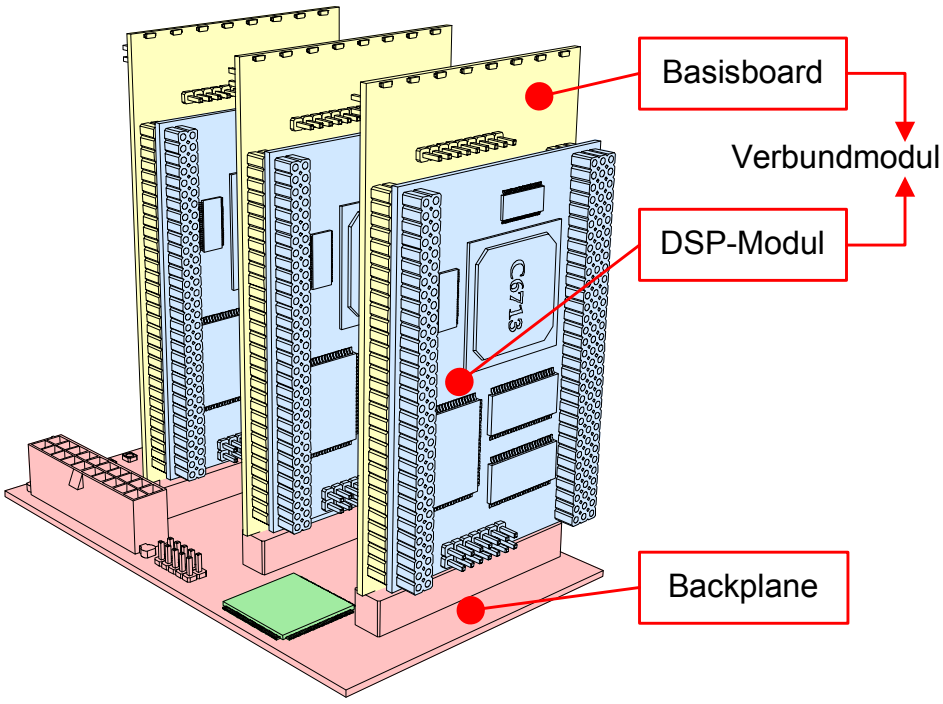



Abb. 2.1.: DSP Kompaktsystem [Hir08]

Jedes DSP-Modul kann mit einem Basisboard zusammengesteckt werden. Dieses Verbundmodul kann über einen entsprechenden Steckplatz mit der Backplane verbunden werden. Die Backplane ermöglicht die elektrische Versorgung von mehreren Verbundmodulen.
Für die Kommunikation zwischen den einzelnen Komponenten sind hardwareseitig verschiedene Strukturen vorgesehen:

- direkte Verbindung zwischen DSP und FPGA (Basisboard) über die EMIF-Schnittstelle
- LVDS-Ringstruktur zwischen den einzelnen Verbundmodulen
- McBSP-Busstruktur zwischen den DSPs der Module
- McBSP-Sternstruktur über einen FPGA-Switch auf der Backplane

### 2.1.1. DSP-Modul D.Module.C6713

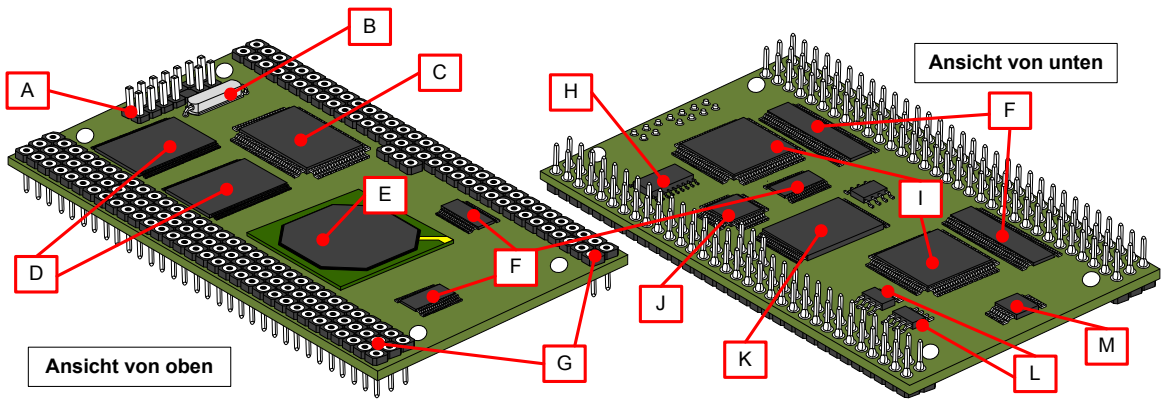
Das D.Module.C6713 von D.SignT verfügt über die moderne CPU TMS320C6713B (kurz: C6713) von Texas Instruments [Tex06], deren Schnittstellen über Steckverbinder am Modul zugänglich sind.

Der C6713 verfügt über folgende Schnittstellen: [D.S04]

- einen seriellen RS232-Port
- zwei Multichannel Buffered Serial Ports (McBSP)
- eine universelle Speicherschnittstelle (EMIF)
- ein Hostport-Interface (HPI)
- 16 (40 bei deaktivierten HPI) frei konfigurierbare GPIO-Signale

Weitere Vorteile zu den zahlreichen Schnittstellen sind:

- interner L2-Cache
- Taktraten bis zu 300 Mhz
- programmierbarer Xilinx CPLD mit 72 Makrozellen




| | |
|---|---|
| A | JTAG-Schnittstelle für das On-Chip-Debugging und die CPLD-Programmierung |
| B | Modulseitiger Taktgeber (Quartz) |
| C | SBSRAM/SSRAM (512 KB) |
| D | 2-mal 16 MByte SDRAM |
| E | Texas Instruments TMS320C6713 (DSP) |
| F | 16-Bit und 32-Bit Bus-Switches |
| G | Externes Bus-Interface (modulares Steckprinzip) |
| H | RS-232 Treiber und Empfänger |
| I | 2-mal Xilinx XC9572XL-TQ100 (CPLD) |
| J | Microcontroller für UART und IRDA |
| K | Flash-Speicher (2 MByte) |
| L | Supervisor Circuits |
| M | Spannungswandler |

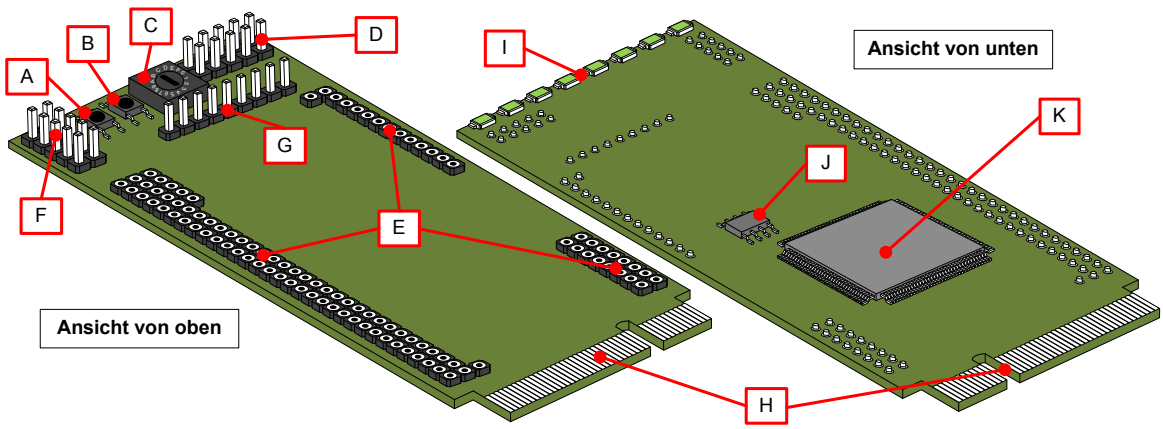
Abb. 2.2.: Das DSP-Modul „D.Module.C6713“ von D.SignT [Mül09]

### 2.1.2. Basisboard

Mit dieser Platine ist die Anbindung des DSP-Moduls an die Backplane über einen Steckkartenverbinder möglich. In einer Erweiterung wird die LVDS-Kommunikationsschnittstelle bereitgestellt. Die notwendige Logik, um eine Kommunikation zwischen EMIF-Schnittstelle des DSP-Moduls und LVDS-Schnittstelle der Backplane herzustellen, befindet sich auf einem FPGA. Über einen ByteBlaster-Anschluss ist dieser frei programmierbar.

Zusätzlich befinden sich auf dem Board:

- ein Reset-Schalter für das gesamte Verbundmodul
- ein Setup-Schalter für die Initialisierung des DSP-Moduls
- acht frei konfigurierbare LEDs



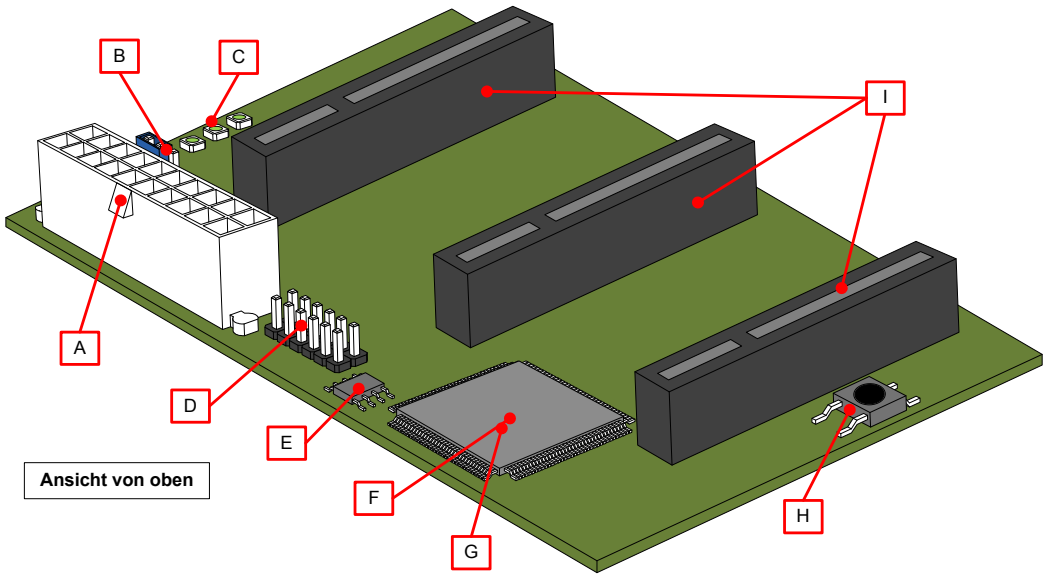

| | |
|---|---|
| A | Setup-Schalter zum Initialisieren des aufgesteckten DSP-Moduls |
| B | Reset-Schalter zum Rücksetzen des gesamten Verbundmoduls |
| C | 4-Bit Kodierschalter für die feste Einstellung einer Hardwareadresse |
| D | Pfostenstecker der seriellen RS232-Schnittstelle |
| E | Bus-Schnittstelle zum DSP-Modul (anwendungsspezifisch genutzte Pins) |
| F | ByteBlaster-Anschluss (derzeit ausschließlich Beschaltung für den AS-Modus) |
| G | Steckkontakte zum Abgreifen der LED-Signale und der Masse (GND) |
| H | Steckkartenverbinder (Bauform PCIe 8x) |
| I | 8 Kontroll-LEDs |
| J | „Serial Programming Device“ EPCS4SI8 für den FPGA |
| K | Altera Cyclone II EP2C8-Q208C7N (FPGA) |

Abb. 2.3.: Das Basisboard des Multi-DSP-Prototyps [Mül09]

### 2.1.3. Backplane

Die Grundplatine des Multi-DSP-Prototyps ist die Backplane. Sie ermöglicht die Stromversorgung der einzelnen Module über PCIe-Steckplätze. Durch einen FPGA wird die sternförmige McBSP-Kommunikation bereitgestellt. Der FPGA übernimmt in diesem Zusammenhang die Switch-Funktion. Die Implementierung der Kommuni-

kationsschnittstelle wurde im Rahmen der Diplomarbeit von Martin Hirsch [Hir08] vorgenommen.




| | |
|---|---|
| A | 3,3V/5V/12V Spannungsversorgung nach ATX-Standard |
| B | Jumper für Soft-Power-Switch des ATX-Netzteils |
| C | LEDs für Spannungskontrolle (POWER-OK/3,3V/5V) |
| D | ByteBlaster-Anschluss (derzeit ausschließlich Beschaltung für den AS-Modus) |
| E | Serielles Programming Device EPCS4SI8 für den FPGA |
| F | Altera Cyclone II EP2C8-T144C7N (FPGA) |
| G | Taktgeber/Quartz (befindet sich auf der Unterseite der Platine) |
| H | Reset-Schalter zum globalen Zurücksetzen des gesamten Systems |
| I | Erweiterungssteckplätze 1, 2 und 3 für Verbundmodule |

Abb. 2.4.: Die Backplane des Multi-DSP-Prototyps (Ansicht von oben) [Mül09]

## 2.2. LVDS-Ring

### 2.2.1. Low Voltage Differential Signaling

Low Voltage Differential Signaling (LVDS) ist ein Schnittstellenstandard für Hochgeschwindigkeits-Datenübertragungen. Im Gegensatz zu den üblichen hohen Spannungen in digitalen Systemen von 5 V oder 3,3 V kommen bei LVDS nur niedrige Spannungen zum Einsatz. Der übliche Spannungshub liegt bei 0,3 V (bei einer absoluten Spannung von 1,2 V). Das heißt, die Signalübertragung ist mit wenig Leistung möglich und starke elektromagnetische Störungen durch hohe Spannungsänderungen und Lade- bzw. Entladeströme werden verhindert.
Bei LVDS-Kommunikationen werden zwei Leitungen (differenzielle Signalübertragung) für den Austausch eines Bits genutzt. Für den Logikpegel ist somit nicht eine Leitungsspannung, sondern die Differenz beider Leitungsspannungen ausschlaggebend. Liegen beide Leitungen nah beieinander, sind die Störungen nahezu gleich ($D+$ und $D-$). Diese Gleichtaktstörungen werden durch den Differenzverstärker enfernt [HH09].

$$v_{diff} = (v_{D+} + v_{noise}) - (v_{D-} + v_{noise}) = v_{D+} - v_{D-}$$

Die Stromquelle auf der Treiberseite erzeugt einen konstanten Strom von 3 mA. Durch einen Abschlusswiderstand von 100 Ω auf der Empfängerseite entstehen Spannungsänderungen von +300 mV und -300 mV, die den Logikpegel repräsentieren. Bei entsprechender Leitungslänge und Kabelqualität können Datenraten im GBit/s Bereich erreicht werden. Abbildung 2.5 zeigt den vereinfachten Aufbau eines LVDS-Übertragungswegs.

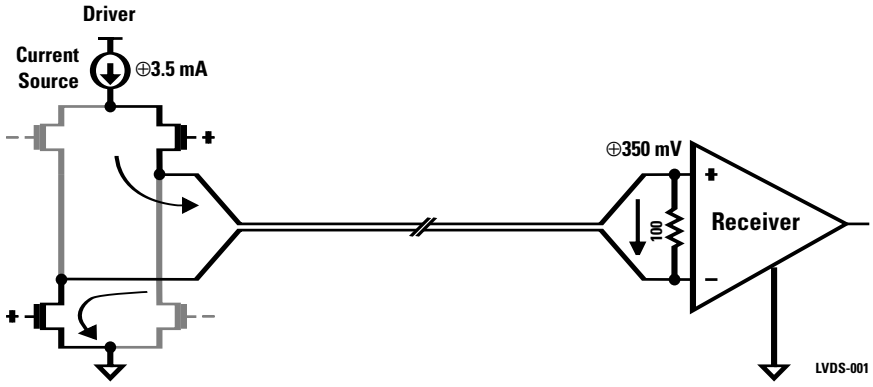



Abb. 2.5.: Vereinfachtes Diagramm eines LVDS-Treibers und LVDS-Empfängers [Nat04]

### 2.2.2. Aufbau

Der LVDS-Ring ist auf der Backplane des Kompaktsystems realisiert (siehe Abbildung 2.6). Durch spezielle Ein- und Ausgänge des FPGAs, die den Übertragungsstandard LVDS unterstützen, wird jeder FPGA mit seinen beiden Nachbarn verbunden. Die abschließende Kopplung des letzten Logikbausteins mit dem ersten erzeugt den Ringschluss des Kommunikationssystems.
Über die EMIF-Schnittstelle ist jedes DSP-Modul mit dem FPGA des Basisboards verbunden. Der FPGA stellt somit die Schnittstelle dar, mit der jeder DSP im LVDS-Ring kommunizieren kann.
Zu beachten ist, dass es sich bei der FPGA-zu-FPGA Verbindung um eine unidirektionale Übertragungsrichtung handelt. Somit müssen Implementierungen für einen unidirektionalen Ring ausgelegt sein.

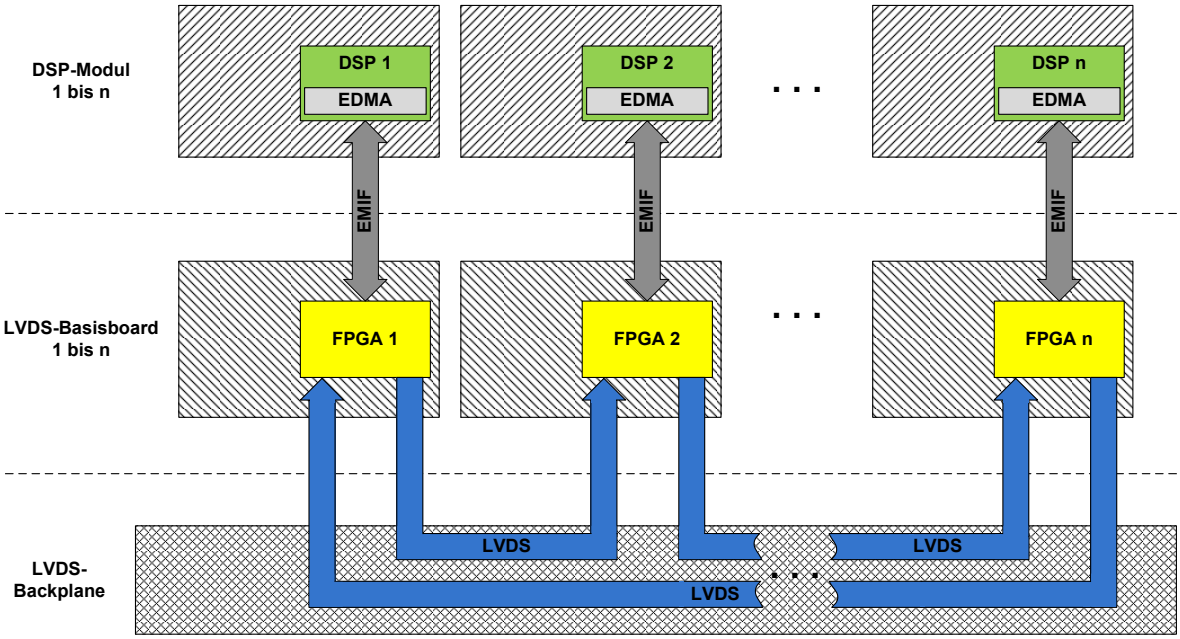



Abb. 2.6.: HW-Struktur: EMIF- und LVDS-basiertes Kommunikationssystem [Mül09]

## 2.3. EMIF-Schnittstelle

Das External Memory Interface (EMIF) ist ein Kommunikationsprotokoll für die Anbindung von externen Speichergeräten an einen Mikroprozessor. Die EMIF-Schnittstelle des DSP TMS320C6713B ermöglicht die Verwendung von synchronem (SBSRAM oder SDRAM) sowie asynchronem Speicher (SRAM oder Flash). Für die Kommunikation im LVDS-Ring soll der DSP über die asynchrone Speicherschnittstelle mit dem Basisboard-FPGA verbunden werden und den Austausch von Paketdaten und Statusinformationen ermöglichen.

### 2.3.1. Asynchrone Schnittstelle

Die asynchrone Schnittstelle bietet durch entsprechende Konfiguration die Möglichkeit, verschiedene Speichertypen zu verwenden. Dazu gehören SRAM, EPROM, Flash-Speicher sowie FPGA- und ASIC-Schaltungen. Abbildung 2.7 zeigt die Anbindung eines SRAM-Bausteins.

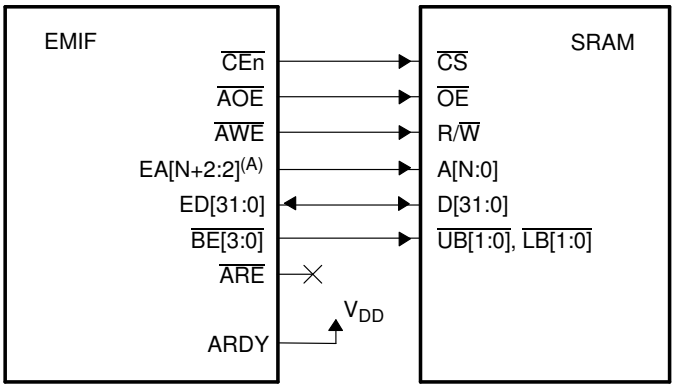



Abb. 2.7.: Anbindung External Memory Interface an SRAM-Baustein [Tex08]

| EMIF Signal | Funktion |
|---|---|
| $\overline{CEn}$ | Command enable - steuert chip select ($\overline{CS}$) des externen Speichers und ist jeweils an ein Kontrollregister im EMIF-Controller gebunden[1] |
| $\overline{AOE}$ | Output enable - aktiv (low) während eines gesamten Lesezugriffes |
| $\overline{AWE}$ | Write enable - aktiv (low) während der STROBE-Periode eines Schreibzugriffes |
| $EA$ | Adressbus - EA[15..0] sind mit dem FPGA verbunden und können von diesem ausgewertet werden |
| $ED$ | Datenbus - 32 Bit breit |
| $\overline{BE}$ | Data output mask - von einigen Speicherbausteinen verwendet, um die Datenausgabe zu maskieren |
| $\overline{ARE}$ | Read enable - aktiv (low) während der STROBE-Periode eines Lesezugriffes |
| $ARDY$ | Ready - dieser Input kann genutzt werden, um einen Wartezustand im Speicherzyklus auszulösen |

Tab. 2.1.: Signalbeschreibung Asynchrone Speicherschnittstelle

[1] Nähere Informationen in [Tex08, Seite 88]

Über ein Konfigurationsregister des C6713 können verschiedene Timing-Parameter eingestellt werden:

**Setup:** Zeit zwischen dem Start eines Speicherzyklus ($\overline{CE}$ low, Adresse gültig) und der Aktivierung eines Lese- oder Schreibzugriffs

**Strobe:** Dauer der Aktivierung eines Lese- ($\overline{ARE}$) oder Schreibzugriffs ($\overline{AWE}$)

**Hold:** Zeit zwischen der Deaktivierung eines Zugriffes und dem Ende des Speicherzyklus (dies kann eine Adressänderung oder die Deaktivierung des $\overline{CE}$ Signals sein)

Diese Parameter beziehen sich auf die Periode des Taktes ECLKOUT. ECLKOUT ist ein Taktausgang[2] für die Synchronisation des EMIF mit dem verwendeten asynchronen Speicher. Die Größe der Parameter ist entscheidend für die Übertragungsgeschwindigkeit zwischen DSP und Speicherbaustein. Allerdings sind diese so zu wählen, dass der Speicher ausreichend Zeit für das korrekte Verarbeiten eines Lese- bzw. Schreibzugriffes hat. Zusätzlich sind minimale Zeiten einzuhalten, die durch den verwendeten DSP vorgegeben sind [Tex08]:

- SETUP $\leq 1/1$ (Lesen/Schreiben)
- STROBE $\leq 1/2$
- HOLD $\leq 0/1$

### 2.3.2. Asynchroner Lesezugriff

Zu Beginn jedes Lesezugriffs (SETUP-Phase) wird $\overline{BE}$ auf einen gültigen Wert gesetzt und die Adresse auf den Adressbus ($EA$) gelegt. Zudem werden $\overline{CE}$ und $\overline{AOE}$ aktiviert ($\overline{CE} = 0$ und $\overline{AOE} = 0$). Mit der STROBE-Phase wird das Read Enable Signal ($\overline{ARE} = 0$) gesetzt. Je nach Länge der STROBE Periode wird dem angeschlossenen Speicherbaustein Zeit gegeben, die korrekten Daten auf den Datenbus zu legen. Im Beispiel Abbildung 2.8 sind das 3 Taktperioden von ECLKOUT.
Unmittelbar vor Beginn der HOLD-Phase werden die Daten vom Datenbus abgerufen,

[2] Taktfrequenz = 100 MHz (bezogen auf den verwendeten DSP TMS320C6713B)

anschließend wird $\overline{ARE}$ zurückgesetzt. Sollen keine weiteren Lesezugriffe erfolgen, wird mit dem Ende der HOLD-Phase $\overline{CE}$ wieder inaktiv.

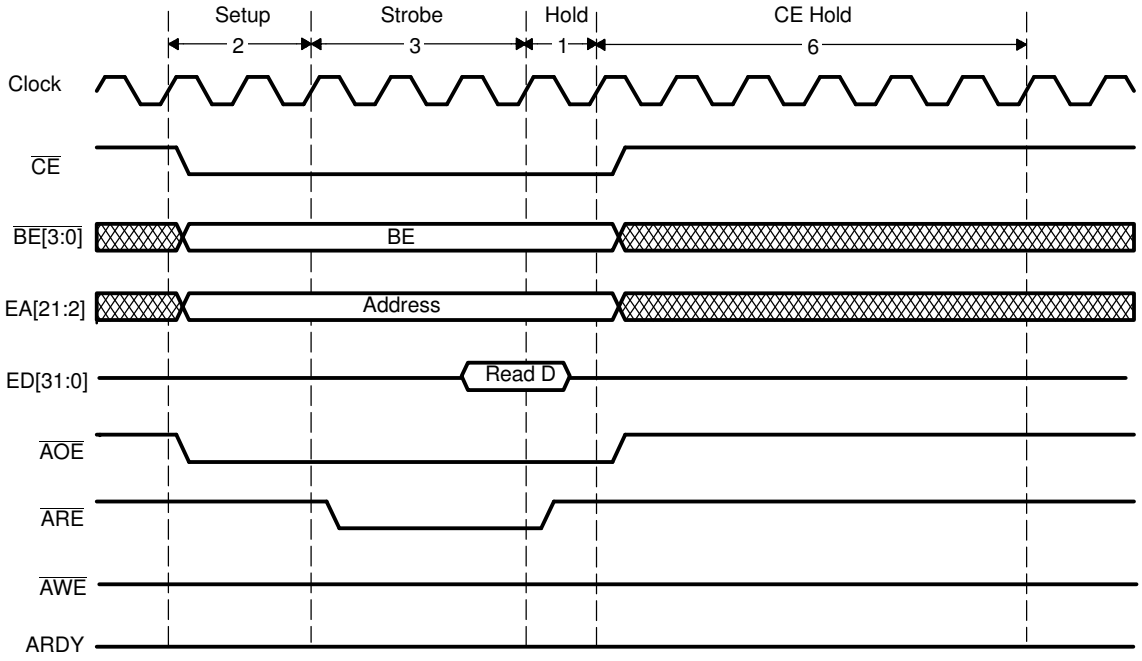



Abb. 2.8.: EMIF Lesezugriff [Tex08, Figure 1-9]

### 2.3.3. Asynchroner Schreibzugriff

In ähnlicher Weise wie der Lesezugriff erfolgt der Schreibzugriff. In Abbildung 2.9 ist der Ablauf des Schreibzugriffes beispielhaft dargestellt.

Die SETUP-Phase startet mit dem Aktivieren von $\overline{CE}$ und dem Ausgeben von gültigen Werten auf Adress- und Datenbus sowie $\overline{BE}$-Signalleitungen.
Zu Beginn der STROBE-Phase wird das Write Enable Signal aktiv ($\overline{AWE} = 0$), welches mit Einleitung der HOLD-Phase deaktiviert wird. Der Schreibzugriff endet mit dem Rücksetzen von $\overline{CE}$ und dem Übergang des Datenbusses in den hochohmigen Zustand (Tristate-Technologie).

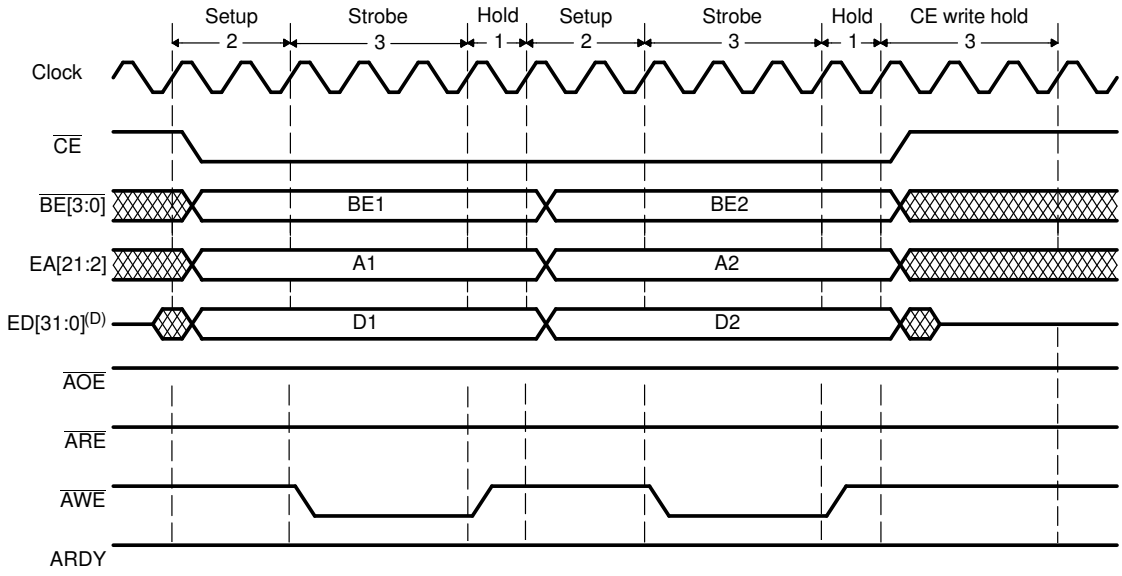



Abb. 2.9.: EMIF Schreibzugriff [Tex08, Figure 1-10]

### 2.3.4. EMIF Register

Wie bereits in Abschnitt 2.3.1 erwähnt, besitzt der EMIF-Controller Register zur Konfigurierung der Schnittstelle. Für die Steuerung des Asynchronen Schreibzugriffes werden das Global Control Register und die CE Space Control Register benötigt.

**Global Control Register (GBLCTL)** Das Globale Kontrollregister enthält Einstellungsmöglichkeiten, die sich auf alle CEx-Speicherbereiche beziehen. Dies beinhaltet die Aktivierung unterschiedlicher Clock-Ausgänge, Buszugriffs-Flags und spezieller Signalleitungen für Handshake-Verfahren mit dem angeschlossenen Gerät.

**CE Space Control Register (CECTL0-3)** Es gibt insgesamt 4 Speicherbereich-Kontrollregister, die mit den externen CE-Signalen ($\overline{CE0}, \overline{CE1}, \overline{CE2}, \overline{CE3}$) korrespondieren. Diese Register beinhalten Informationen über den angeschlossenen Speichertypen und die verwendeten Timing-Parameter für SETUP-, STROBE- und HOLD-Periode.

Abbildung 2.10 und 2.11 zeigen den Aufbau der Register. Nähere Informationen zu den Parametern findet man im Reference Guide von Texas Instruments [Tex08, Seite 86-92]

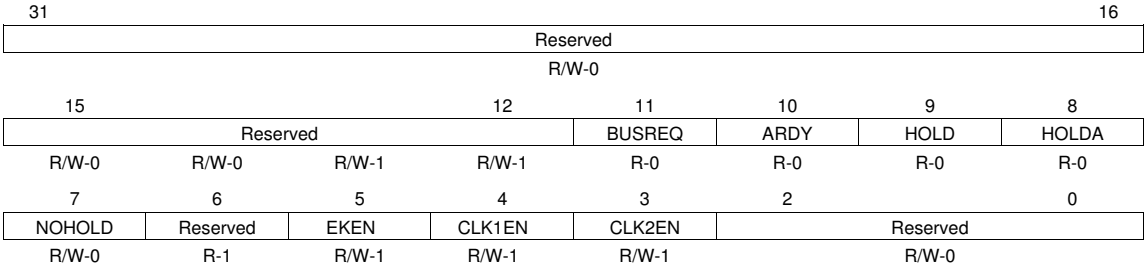



Abb. 2.10.: EMIF Global Control Register (GBLCTL) [Tex08, Figure 3-15]

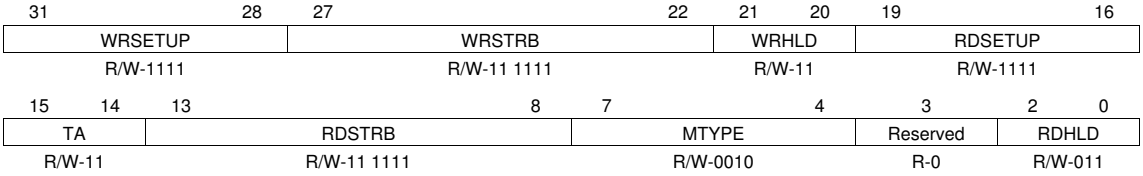



Abb. 2.11.: EMIF CE Space Control Register (CECTL0-3)[Tex08, Figure 3-16]

# 3. Entwurfsmethodik

## 3.1. Modellbasierter Entwurf

Durch die zunehmende Komplexität elektronischer Systeme und Implementierungen ist es unumgänglich, neue Methoden des Entwurfes zu entwickeln und zu nutzen. Die modellbasierte Entwurfsmethodik bietet durch ihr Abstraktionsprinzip Vorteile in der Strukturierung und Umsetzung von Problemstellungen.
Im Gegensatz zur klassischen Vorgehensweise bei der Implementierung mit hardwarenahen Programmiersprachen werden im modellbasierten Entwurf Prozesse grafisch aufbereitet und sind somit leichter nachzuvollziehen.
Durch die zunehmende Weiterentwicklung von Entwurfswerkzeugen in diesem Bereich gewinnt ein weiterer Vorteil zunehmend an Bedeutung – aktuelle Werkzeuge ermöglichen die Verifizierung und Validierung während des gesamten Entwurfsprozesses. Durch Computersimulationen ist die Erkennung und Beseitigung von Fehlern frühzeitig möglich.
Die letztendliche Umsetzung der Modelle in eine höhere oder hardwarenahe Programmiersprache erfolgt mittels Generatoren. Diese parametrisierbare, generative Codeerzeugung beschleunigt den Entwurfsprozess und vermindert Fehler in der Implementierung.

## 3.2. Entwurfswerkzeuge

In den folgenden Abschnitten wird die verwendete Toolchain näher erläutert. Diese hat sich in den Vorarbeiten ([Mül09], [Sch10]) bewährt und ermöglicht eine effektive Vorgehensweise beim hardwarenahen Entwurf.

### 3.2.1. Matlab/Simulink

Der modellbasierte Entwurf spielt im Fachgebiet Rechnerarchitektur eine große Rolle. Neben dem Modellierungs- und Simulationswerkzeug MLDesigner wird Matlab mit der Erweiterung Simulink für die Entwicklung komplexer eingebetteter und vernetzter Systeme verwendet.
In Verbindung mit der Erweiterung *Stateflow* bietet Matlab/Simulink vergleichbare Entwurfsmöglichkeiten wie MLDesigner. Durch die grafische Umgebung ist ein interaktiver Entwurf der Hardware übersichtlich möglich. Eine eigene Bibliothek enthält Funktionsblöcke, mit denen das System modelliert werden kann. Durch eine zeitgesteuerte (*Simulink*) und ereignisorientierte (*Stateflow*) Simulation kann das Modell während des Entwurfs verifiziert werden.
Der Vorteil von Matlab liegt bei der Codegenerierung. Durch zusätzliche Plugins[1] können die erstellten Modelle in eine Hardwarebeschreibung oder eine Software-Programmiersprache umgesetzt werden. Weitere Vor- und Nachteile dieser Entwurfsmethodik wurden in der Arbeit von Michael Müller [Mül09] näher erläutert.

### 3.2.2. HDL Coder

Simulink-Modelle können mit dem Plugin HDL Coder in eine Hardwarebeschreibungssprache[2] (HDL) überführt werden.
Für eine korrekte Umsetzung ist es notwendig, dass das zu übersetzende System ein zeitdiskretes synchrones Modell, bestehend aus HDL Coder-konformen Elementen, ist. Der HDL Coder bietet hierfür eine eigene Bibliothek, die mit dem Matlab-Befehl *hdllib* aufgerufen werden kann. Diese enthält neben den konformen Elementen der Simulink-Bibliothek auch zusätzliche Blöcke für Bit-Operationen oder Speicherzugriffe.
Sollten Teile des Modells nicht durch den Codegenerator umgesetzt werden können, erhält der Nutzer eine Fehlermeldung mit dem Verweis auf die entsprechende Stelle im Modell. Nach dem erfolgreichen Generierungsprozess können die strukturierten HDL-Dateien in einer Simulationsumgebung oder einem Synthesetool weiterverwendet werden.

[1] HDL Coder, Stateflow Coder, Real-Time Workshop
[2] VHDL oder Verilog

Zusätzlich ermöglicht der HDL Coder das Integrieren von Daten aus der Simulink-Simulation in eine Testbench, die dann beispielsweise von der Simulationsumgebung *ModelSim* verwendet werden kann.

### 3.2.3. ModelSim

Als Zwischenschritt vor der Synthese mit Quartus hat sich die Simulation mit ModelSim bewährt. Es ist vorgekommen, dass Teile des Simulink-Modells nicht im VHDL-Quelltext umgesetzt wurden und vom HDL Coder keine Fehlermeldung erfolgte.
ModelSim simuliert das System auf RTL- und Gate-Ebene sowohl zeitgenau als auch taktsynchron. Mit der Möglichkeit des HDL Coders, eine Testbench zu erstellen, kann der generierte VHDL-Code komfortabel überprüft werden.

### 3.2.4. Quartus II

Quartus II ist eine Software der Firma ALTERA, die eine komplette Entwicklungsumgebung für FPGA-basierte Systeme bietet. Abbildung C.1 zeigt den typischen Ablauf einer Designentwicklung.

Neben einem kompletten Durchlauf ermöglicht es der modular aufgebaute Compiler, die Module einzeln zu verwenden. Das Design wird zunächst auf logische und schaltungstechnische Fehler überprüft und abschließend simuliert, um Verbindungen und I/O Pins zu überprüfen.
Quartus II bietet für die LVDS-Anbindung konfigurierbare IP-Cores. Diese werden mit dem entwickelten LVDS-Knoten verbunden, um das Gesamtsystem zu synthetisieren. Durch die Protokolle des Compilers kann abschließend die Synthese hinsichtlich der Timings und verwendeten internen Ressourcen überprüft werden.

# 4. Vorarbeiten

## 4.1. LVDS-Knoten

### 4.1.1. Funktionsweise

Im Rahmen der Bachelorarbeit von Raik Schulze ist bereits ein Kommunikationsprotokoll für den LVDS-Ring entstanden. Die dafür notwendige Logik, im Folgenden als LVDS-Knoten bezeichnet, ist auf dem FPGA des Basisboards implementiert und stellt das zentrale Bindeglied zwischen DSP und Kommunikationssystem dar.
Folgende Funktionen sind im LVDS-Knoten enthalten:

- Initialisierung des LVDS-Rings
- Interpretieren und entsprechendes Bearbeiten von Datenpaketen im LVDS-Ring
- Schnittstelle für eine indirekte Anbindung an den DSP
- Fehlererkennung und -korrektur in Paketen

Da das Prototypen-Board noch nicht hinsichtlich der Anzahl und Häufigkeit auftretender Übertragungsfehler analysiert werden konnte, wurde in die FPGA-Logik eine Fehlerkorrektur von 1-Bit Fehlern und eine Fehlererkennung von 2-Bit Fehlern integriert.
Zur Fehlerkorrektur und -erkennung wird der erweiterte separierte (31,5) Hamming-Code [Wik09] verwendet. Mit einer zusätzlichen ungeraden Parität über den gesamten Nachrichteninhalt können Trivial-Fehler[1] identifiziert werden.

[1] gesamtes Datenwort besteht nur aus Nullen bzw. Einsen

### 4.1.2. Paketaufbau

Das Kommunikationsprotokoll sieht Datenworte, im Folgenden mit Frame bezeichnet, mit einer Breite von 32 Bit vor. Eine geschlossene Sequenz von Frames wird als Paket bezeichnet. Ein Paket besteht aus einem Headerframe und maximal 31 Datenframes. Der Aufbau dieser zwei Frametypen ist Tabelle 4.1 und Tabelle 4.2 zu entnehmen.

| | HeaderID | SenderID | ReceiverID | Data | Length | Parity |
|---|---|---|---|---|---|---|
| Bitposition | 31-27 | 26-22 | 21-17 | 16-11 | 10-6 | 5-0 |

Tab. 4.1.: Aufteilung Headerframe [Sch10]

| | Data | Parity |
|---|---|---|
| Bitposition | 31-6 | 5-0 |

Tab. 4.2.: Aufteilung Datenframe [Sch10]

Der Headerframe stellt wichtige Informationen für die Steuerung des Kommunikationsprotokolls bereit. Datenframes enthalten die übermittelten Informationen für den DSP.

Grundsätzlich werden sechs verschiedene Header-IDs unterschieden[2]. Für diese Arbeit sind die IDs 4 und 5 relevant (siehe Abschnitt 5.1.1). Header-ID 4 kennzeichnet einen Idle-Frame der verschickt wird, wenn keine weiteren Datenpakete vorliegen. Header-ID 5 kennzeichnet den Anfang eines Datenpakets, die nachfolgenden Datenframes[3] enthalten die zu übertragenden Informationen.

Durch die Mechanismen zur Fehlerkorrektur und die damit verbundenen Paritätsbits in Daten- und Headerframes stehen dem DSP 26 Bits zur Verfügung, die mit relevanten Informationen gefüllt werden können. Da ein Datenpaket (HeaderID=5) eine maximale Länge von 31 Datenframes besitzen kann, ergeben sich höchstens $31 \cdot 26 = 806$ Informationsbits für die Übertragung zwischen zwei DSPs.

[2] näheres in [Sch10, Tabelle 5.3]
[3] Anzahl der Datenframes ist abhängig vom Length-Parameter im Headerframe.

### 4.1.3. Puffer

Der LVDS-Knoten enthält zwei interne Speicher - den Sendepuffer und den Empfangspuffer.
Der Sendepuffer dient der Zwischenspeicherung der zu übertragenden Datenpakete. Diese werden vom DSP übermittelt und solange im Zwischenspeicher gehalten, bis eine Übertragung im LVDS-Ring möglich ist.
In den Empfangspuffer werden die Datenpakete abgelegt, die vom LVDS-Knoten empfangen werden. Diese können anschließend ausgelesen und an den DSP weitergeleitet werden.

### 4.1.4. Schnittstelle

Für den Zugriff auf Sende- und Empfangsfunktion sowie weitere Statusinformationen des LVDS-Knotens ist eine Schnittstelle vorgesehen. Abbildung 4.1 und Tabelle 4.3 zeigen die Signale, die von der Schnittstelle zur Verfügung gestellt werden.

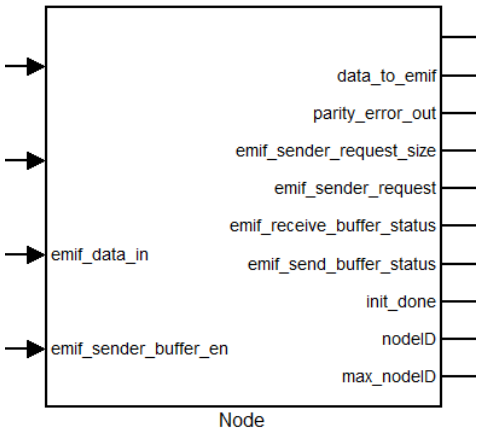



Abb. 4.1.: Schnittstelle des LVDS-Knotens

| Signal | Funktion |
|---|---|
| INPUT | |
| emif_data_in | Ermöglicht den Zugriff auf den Sendepuffer; Erkennt der LVDS-Knoten an diesem Eingang eine Header-ID ungleich 4, so werden in den nachfolgenden Takten die Datenframes erwartet[4]. |
| emif_sender_buffer_en | Enable-Signal für den Empfangspuffer; Ist dieses auf 1 gesetzt, so wird mit jedem Takt ein Frame aus dem Empfangspuffer über *data_to_emif* ausgegeben. |
| OUTPUT | |
| data_to_emif | Enthält die Daten aus dem Empfangspuffer und das dazugehörende Error-Bit, wenn *emif_sender_buffer_en* aktiv ist; |
| parity_error_out | Extra Signalausgang für das Error-Bit; |
| emif_sender_request_size | Übermittelt die Größe eines neu empfangenen Datenpakets; |
| emif_sender_request | Ist für einen Takt aktiv, wenn ein neues Datenpaket empfangen wurde; |
| emif_receive_buffer_status | Füllstand des Empfangspuffers; |
| emif_send_buffer_status | Füllstand des Sendepuffers; |
| init_done | Signalisiert die abgeschlossene Initialisierung des LVDS-Rings; |
| nodeID | Aktuelle Knoten-ID; |
| max_nodeID | Maximale Knoten-ID im LVDS-Ring; |

Tab. 4.3.: Schnittstelle des LVDS-Knotens

[4]Die Anzahl der erwarteten Datenframes wird durch den Längenparameter im Headerframe definiert.

# 5. Konzept

## 5.1. Konzeption der EMIF-Bridge

Ziel dieser Arbeit ist es, die Verbindung zwischen LVDS-Knoten und DSP zu realisieren. Die dafür notwendige „EMIF-Bridge“ muss folgende funktionale Aufgaben erfüllen:

- Senden von Datenpaketen des DSPs
- Empfangen und Weiterleiten von Datenpaketen an den DSP
- Zugriff auf Status- und Kontrollregister
- Generierung von Interrupts bei bestimmten Systemereignissen

Die EMIF-Bridge kann in vier seperat arbeitende Teile zerlegt werden – Signallogik, Sendelogik, Empfangslogik und Interruptlogik. Die Signallogik wird in den folgenden Abschnitten nicht einzeln aufgeführt. Die zwei relevanten Teile für das Erkennen von Schreib- und Lesezugriffen des EMIFs werden in die Beschreibungen der Sende- bzw. Empfangslogik einbezogen.
Desweiteren sollen die Komponenten der EMIF-Bridge folgende technische Anforderungen erfüllen sollen:

- geringe Latenzen
- erreichen einer Taktrate von 100 MHz
- leichte Erweiterbarkeit
- angemessene Fehlertoleranz

### 5.1.1. Sendelogik - Datenübertragung von DSP zum LVDS-Knoten

Um Daten im LVDS-Ring übermitteln zu können, steht dem DSP die EMIF-Schnittstelle zur Verfügung (siehe Abschnitt 2.3), die über einen Schreibzugriff die Verbindung zum FPGA herstellt.
Die Vermittlung zwischen den Protokollen von EMIF-Schnittstelle und LVDS-Knoten übernimmt die EMIF-Bridge.

Der LVDS-Knoten erwartet ein komplettes Paket taktsynchron am Eingang *emif_data_in*. Soll kein Datenpaket übermittelt werden, so muss an diesem Eingang ein Headerframe mit der Header-ID 4 anliegen. Das heißt, wird eine Header-ID 5 an diesem Eingang vom LVDS-Knoten erkannt, werden entsprechend dem Längenparameter des Headerframes in den nachfolgenden Takten die Datenframes erwartet.
Diese taktsynchrone Datenübertragung kann durch das EMIF nicht ermöglicht werden, da hier zusätzliche Takte für das Setzen der Steuersignale benötigt werden. Aus diesem Grund ist eine zusätzliche Pufferung des vollständigen Pakets in der EMIF-Bridge nötig. Erst wenn ein Paket vollständig im Puffer abgespeichert wurde, kann eine protokollgerechte Übertragung an den LVDS-Knoten sichergestellt werden.

Zusätzlich zu dem Übergeben von Paketen an den LVDS-Knoten soll mit einem Schreibzugriff auch die Steuerung der EMIF-Bridge möglich sein. Zu diesem Zweck sind zwei Kontrollregister vorgesehen (siehe Abschnitt 5.1.5).
Den schematischen Aufbau der Sendelogik[1] zeigt Abbildung 5.1.

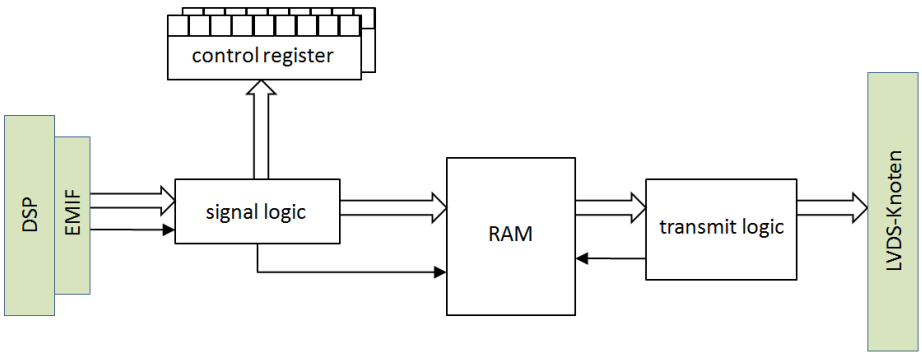



Abb. 5.1.: Konzept Sendelogik

[1] Dicke Pfeile kennzeichnen die Datenleitungen für die Frames. Dünne Pfeile kennzeichnen Steuersignale.

Die Signallogik (*signal logic*) kontrolliert die anliegenden EMIF-Signale. Liegt ein Schreibzugriff vor, werden die Daten entsprechend der Adresse auf dem Adressbus an ein Kontrollregister (*control register*) oder an den internen Zwischenspeicher weitergeleitet.
Die Übermittlungslogik (*transmit logic*) übergibt das Paket an den LVDS-Knoten, sobald alle Datenframes vollständig eingetroffen sind. Liegen keine Daten vor, wird ein Idle-Frame an den LVDS-Knoten weitergeleitet.

Da der LVDS-Knoten über einen eigenen Sendepuffer verfügt, kann dieser als FIFO-Speicher betrachtet werden. Da eine taktsynchrone Datenübermittlung von der EMIF-Bridge garantiert wird, können Pakete, sofern keine höher priorisierte Datenübertragung (LVDS-Eingang zu LVDS-Ausgang) vorliegt, ohne weitere Zwischenpufferung in den LVDS-Ring übermittelt werden.

### 5.1.2. Empfangslogik - Datenübertragung vom LVDS-Knoten zum DSP

Ein Lesezugriff des EMIF kann zwei unterschiedliche Gründe haben:

- das Auslesen eines Statusregisters der EMIF-Bridge (siehe Kapitel 5.1.4)
- das Abrufen von empfangenen Datenpaket aus dem LVDS-Knoten

Beide Zugriffe können durch unterschiedliche Adressen auf dem Adressbus differenziert werden. Der schematischen Aufbau der Empfangslogik[2] ist in Abbildung 5.2 dargestellt.

[2]Dicke Pfeile kennzeichnen die Datenleitungen für die Frames. Dünne Pfeile kennzeichnen Steuersignale.

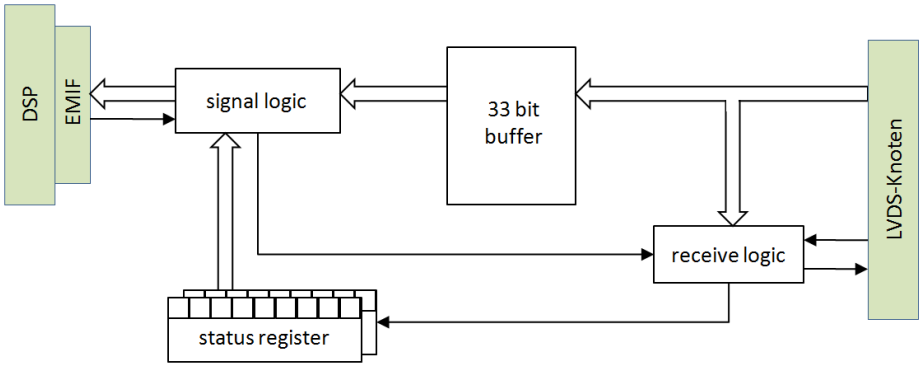



Abb. 5.2.: Konzept Empfangslogik

Die Signallogik (*signal logic*) wertet die EMIF-Signale hinsichtlich eines Lesezugriffs aus und leitet die angeforderten Daten entsprechend weiter.
Für das Übertragen von Datenpaketen zwischen LVDS-Knoten und DSP müssen bestimmte Protokollvorschriften eingehalten werden. Der LVDS-Knoten signalisiert ein vollständig eingetroffenes Paket mittels des Signals *emif_sender_request*. Da dieses Signal nur einmal pro Paket aktiv wird, ist es nötig, die erhaltenen Requests in der EMIF-Bridge zu zählen und abzuspeichern (Paketzähler).
Für das Übermitteln von Frames des LVDS-Knotens an den DSP ist in der EMIF-Bridge ein Zwischenspeicher vorgesehen. Dieser beinhaltet genau einen Frame und ermöglicht das zeitnahe Übertragen an den DSP. Anschließend wird, sofern vorhanden, erneut ein Frame aus dem LVDS-Knoten im Zwischenspeicher abgelegt. Mit dem Paketzähler und einem zusätzlichen Zähler für die Paketlänge kann eine sichere Aussage darüber getroffen werden, ob der LVDS-Knoten weitere Frames bereitstellen kann.

### 5.1.3. Interruptlogik

Ein wichtiger Teil der Kommunikation zwischen DSP und EMIF-Bridge ist die Interruptlogik. Durch das Senden einer Unterbrechungsanforderung ist es der EMIF-Bridge möglich, eine Interrupt-Service-Routine (ISR) auszulösen und den Prozessor über bestimmte Ereignisse zu informieren.

Da der DSP nur 4 externe Interruptquellen unterstützt, wird aus ressourcensparenden Gründen nur eine externe Interruptleitung von der EMIF-Bridge genutzt. Um trotzdem mehrere Schlüsselereignisse zu unterscheiden, ist es nötig, die unterschiedlichen Interrupt-Ereignisse in einem Statusregister abzulegen.

Zur Vorbeugung eines Interruptverlustes durch eine fehlerhafte Signalübertragung an den DSP sollen die Interrupts nicht flanken- sondern pegelgesteuert realisiert werden. Dies erfordert eine Bestätigung des DSPs, um die Interruptleitung zurückzusetzen (siehe Abschnitt 5.1.5). Erst wenn diese Bestätigung erfolgt ist, kann die EMIF-Bridge eine neue Unterbrechungsanforderung senden und ein Verlust eines Interrupts während der Abarbeitung der Interrupt Service Routine ist ausgeschlossen.
Für die Implementierung sind zwei unterschiedliche Typen der Bestätigung vorgesehen. Durch eine harte Bestätigung (*hard acknowledgement*) soll die Interruptleitung zurückgesetzt werden. Anschließend kann erneut ein Interrupt von der EMIF-Bridge ausgelöst werden.
Durch eine weiche Bestätigung (*soft acknowledgement*) werden die nächsten anstehenden Interrupt-Ereignisse in das Statusregister geladen, die Interruptleitung verbleibt aber in ihrem aktuellen Zustand. Dadurch ist es dem DSP möglich, weitere Interrupt-Ereignisse abzuarbeiten, ohne die ISR zu verlassen.

Der schematische Aufbau der Interruptlogik ist in Abbildung 5.3 dargestellt[3].

Die Interrupt-Request-Logik generiert Unterbrechungsanfragen für folgende Ereignisse:

- ein Datenpaket ist vollständig im LVDS-Knoten eingetroffen
- ein Kommunikations-Parameter wurde geändert (*Node-ID*, *Max-ID* oder *init_done*)
- eine eingestellte Sendepuffer-Grenze wurde unterschritten
- eine eingestellte Empfangspuffer-Grenze wurde überschritten

Die generierten Interrupt-Anfragen werden zunächst maskiert (*Interrupt Mask Register* (*IMR*)). Dies ermöglicht es dem Anwender, bestimmte Interrupt-Ereignisse zu deakti-

[3]Dicke Pfeile kennzeichnen die generierten Interrupt-Ereignisse. Dünne Pfeile kennzeichnen Steuersignale.

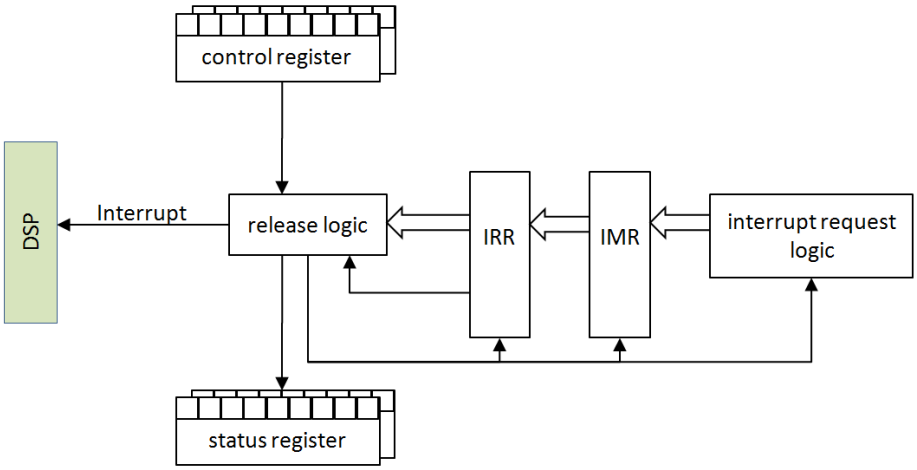



Abb. 5.3.: Konzept Interruptlogik

vieren. Die maskierten Interrupt-Anfragen werden in einem Register (*Interrupt Request Register* (*IRR*)) zwischengespeichert.
Die Freigabelogik (*release logic*) übernimmt die Steuerung der Interruptleitung entsprechend des zuvor beschriebenen Handshake-Verfahrens und speichert das aktuelle Interrupt-Ereignis in ein spezielles Statusregister.

### 5.1.4. Statusregister

Die EMIF-Bridge stellt zwei Statusregister zur Verfügung – das Globale Statusregister und das Interrupt-Statusregister.

#### Globales Statusregister (GLST)

Das Globale Statusregister stellt allgemeine Informationen der Sende- und Empfangslogik zur Verfügung. Dies beinhaltet die aktuellen Netzwerkparameter (*Node-ID*, *Max-ID* und *init_done*), Informationen zum Frame im Zwischenspeicher sowie die Füllstände von Empfangs- und Sendepuffer des LVDS-Knotens.
Der Aufbau des Registers kann Abbildung 5.4 und Tabelle 5.1 entnommen werden.

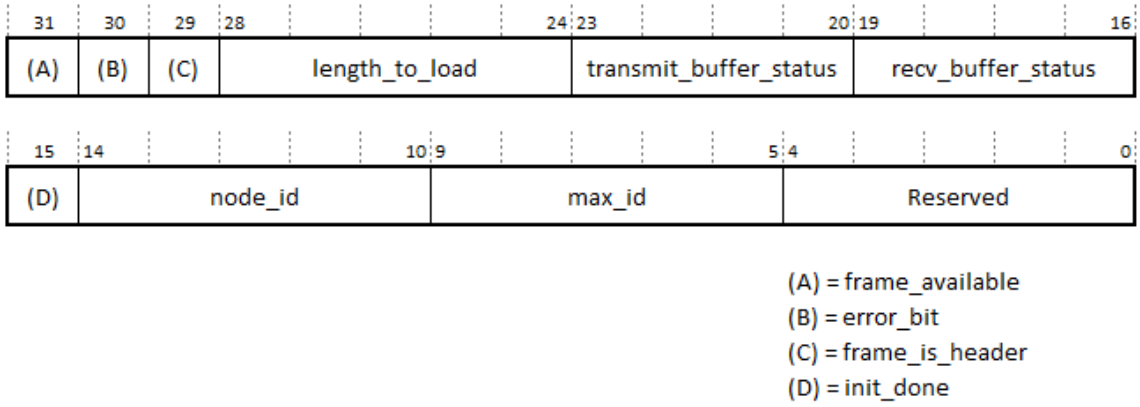



Abb. 5.4.: Globales Statusregister

| Bezeichnung | Position | Beschreibung |
|---|---|---|
| frame_available | 31 | Enthält eine 1, sobald ein Frame abrufbereit im Zwischenspeicher liegt; |
| error_bit | 30 | Signalisiert, ob der zwischengespeicherte Frame defekt ist (error_bit=1); |
| frame_is_header | 29 | Enthält eine 1, wenn der Frame im Zwischenspeicher ein Header ist; |
| length_to_load | 28..24 | Noch zu ladende Länge des aktuellen Datenpakets; |
| transmit_buffer_status | 23..20 | Füllstand des LVDS-Knoten-Sendepuffers; |
| recv_buffer_status | 19..16 | Füllstand des LVDS-Knoten-Empfangspuffers; |
| init_done | 15 | Signalisiert, ob die Initialisierung des Knotens abgeschlossen wurde; |
| node_id | 14..10 | Knoten-ID im LVDS-Ring; |
| max_id | 9..5 | Maximale Knoten-ID im LVDS-Ring; |

Tab. 5.1.: Belegung des Globalen Statusregisters

### Interrupt-Statusregister (IRQST)

Das Interrupt-Statusregister enthält Informationen zur aktuellen IRQ-Maskierung, dem Status des Interrupt-Request-Registers und den aktuellen Interruptereignissen.

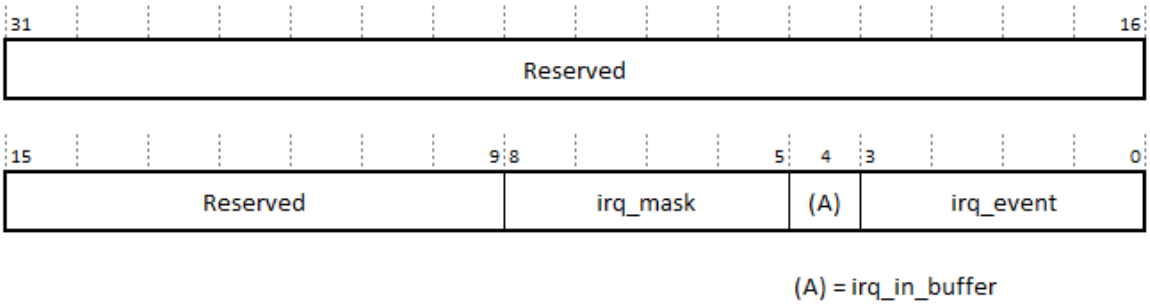



Abb. 5.5.: Interrupt Statusregister

| Bezeichnung | Position | Beschreibung |
|---|---|---|
| irq_mask | 8..5 | aktuelle IRQ-Maske (1=Interrupt aktiviert, 0=Interrupt deaktiviert):<br>1 – – – b Paket empfangen (*package_recv*)<br>– 1 – – b Änderung von Kommunikationsparametern (*node_change*)<br>– – 1 – b Grenze des Sendepuffers unterschritten (*sendbuffer_border*)<br>– – – 1 b Grenze des Empfangspuffers überschritten (*recvbuffer_border*) |
| irq_in_buffer | 4 | Signalisiert, ob weitere IRQs im Puffer (IRR) liegen; |
| irq_event | 3..0 | Aktuelles IRQ-Ereignis (1=Ereignis aktiv, 0=Ereignis inaktiv); Belegung siehe IRQ-Maske; |

Tab. 5.2.: Belegung des Interrupt-Statusregisters

Die IRQ-Ereignisse können beliebig kombiniert sein. Beispielsweise bedeutet $1001b$, dass ein Paket empfangen und die Grenze des Empfangspuffers überschritten wurde.

### 5.1.5. Steuerregister

Über Steuerregister können spezielle Funktionen der EMIF-Bridge genutzt werden. Die EMIF-Bridge enthält ein Globales Steuerregister und ein Interrupt-Steuerregister.

#### Globales Steuerregister (GLCTRL)

Das Globale Steuerregister ermöglicht das Zurücksetzen der drei Teilkomponenten der EMIF-Bridge:

- Sendelogik
- Empfangslogik
- Interruptlogik

Der Aufbau des Globalen Steuerregisters wird in Abbildung 5.6 gezeigt.

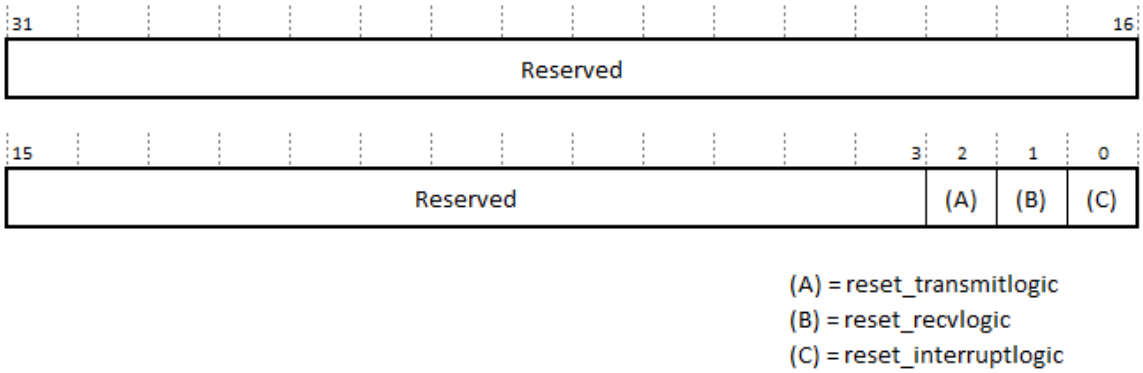



Abb. 5.6.: Globales Kontrollregister

Es kann sowohl ein Reset der einzelnen Teilkomponenten, als auch eine Kombination von ihnen durchgeführt werden. Beispielsweise erfolgt durch $GLCTRL = 111b$ ein Rücksetzen der gesamten EMIF-Bridge.

Die Bedeutungen der Resets und ihre Auswirkungen kann Abschnitt 7.1.2 entnommen werden.

### Interrupt-Steuerregister (IRQCTRL)

Das Interrupt-Steuerregister ermöglicht den Zugriff auf fünf Funktionen:

- Reset der Interruptlogik
- Bestätigung für das Handshake-Verfahren
- IRQ-Maskierung
- Setzen der Sendepuffer-Grenze
- Setzen der Empfangspuffer-Grenze

Abbildung 5.7 zeigt den Aufbau des Interrupt-Steuerregisters.

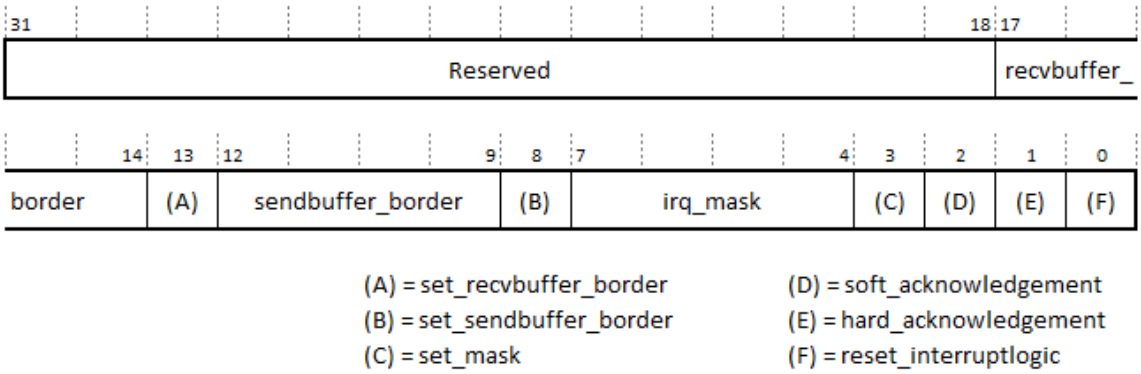



Abb. 5.7.: Interrupt Kontrollregister

Die Bedeutung für *hard_acknowledgement* und *soft_acknowledgement* wurde in Abschnitt 5.1.3 näher erläutert.
Um eine neue Empfangspuffer-Grenze, Sendepuffer-Grenze oder Interrupt-Maske festzulegen, ist jeweils ein *set*-Bit vorgesehen. Erst wenn dieses gesetzt ist, werden die übermittelten Daten abgespeichert. Die Belegung der Interrupt-Maske (*mask*) kann Tabelle 5.2 entnommen werden.

## 5.2. Konzept der Simulation

Für die Simulation des LVDS-Knotens mit verschiedenen Fehlerszenarien und Lastbedingungen in Matlab/Simulink ist es notwendig, eine flexibel konfigurierbare Simulationsumgebung zu erstellen. Abbildung 5.8 zeigt die umgesetzte Struktur der Simulationsumgebung.

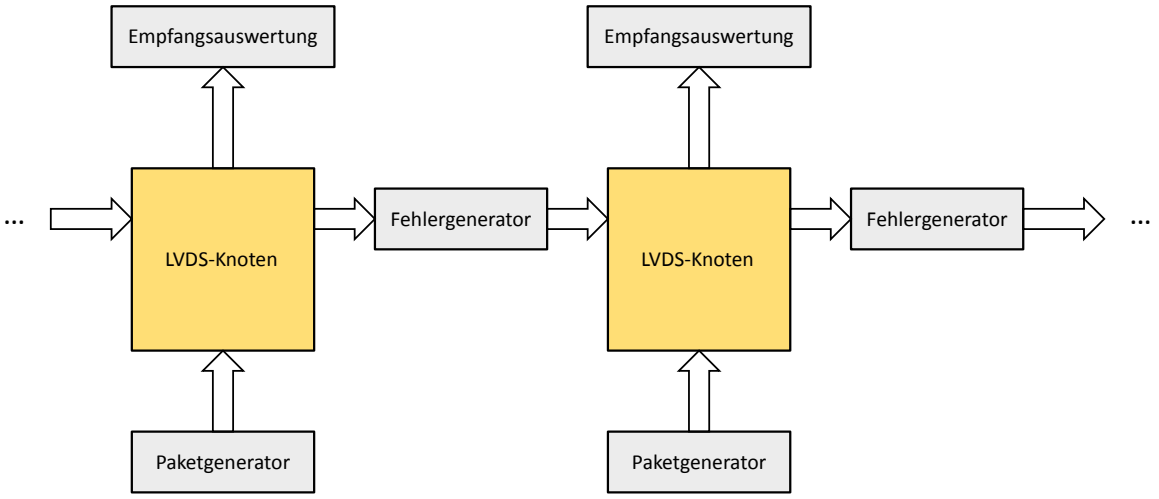



Abb. 5.8.: Aufbau Simulationsumgebung des Modelltests

Die folgenden Kapitel beschreiben die drei elementaren Simulationskomponenten:

1. Paketgenerator
2. Fehlergenerator
3. Empfangsauswertung

Zusätzlich zu diesen drei Komponenten wird eine Steuerlogik benötigt, die die Blockparameter entsprechend des gewünschten Simulationsverhaltens anpasst.

### 5.2.1. Paketgenerator

Um Pakete für den simulierten LVDS-Knoten zu erstellen, wird ein Generator benötigt. Dieser Generator soll folgende Eigenschaften besitzen, um verschiedene Lastprofile zu erzeugen:

- zufällige Paketlänge in einem einstellbaren Bereich
- zufällige Länge für Pausen zwischen generierten Paketen
- abwechselndes Versenden von Paketen an alle Knoten im Ring
- Start-/Stop-Möglichkeit für den Generator

In Abbildung 5.9 ist der Ablauf der Paketgenerierung an einem Beispiel dargestellt. Vor der Generierung eines Datenpakets mit zufälligem Inhalt (Paket-Phase) liegt die Übertragungs-Phase. Diese simuliert die Übertragung zwischen DSP und EMIF-Bridge und ist ein konfigurierbares Vielfaches der nachfolgenden Paketlänge. Der Faktor ist abhängig von den verwendeten Timing-Parametern des EMIF-Controllers. Wie im Beispiel verwendet, kann er, um die Simulationszeit zu verringern, auch auf 1 gesetzt werden.
Anschließend an die Paket-Phase erfolgt die Idle-Phase. Diese simuliert mit einer zufälligen Länge die Abarbeitungszeit des DSPs, in der keine Pakete verschickt werden.

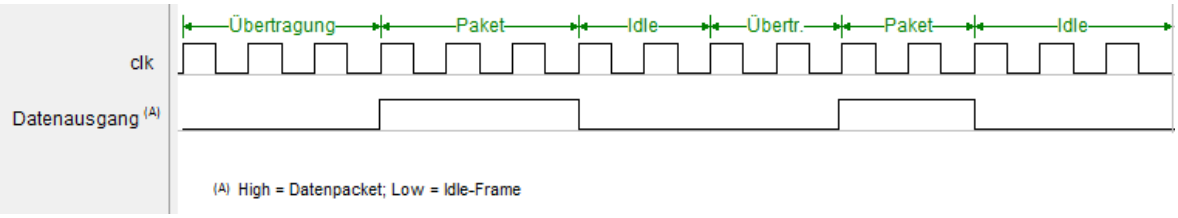



Abb. 5.9.: Ablauf der Paketgenerierung

Der Inhalt eines Datenpakets besteht aus Pseudozufallszahlen (vgl. [Sch10, Seite 13]). Dies hat den Vorteil, dass der Inhalt zu jedem Zeitpunkt nachvollzogen und überprüft werden kann.

### 5.2.2. Fehlergenerator

Der Fehlergenerator ist für das Einschleusen von Bitfehlern in der Nachrichtenübertragung zwischen zwei LVDS-Knoten zuständig.
Im Vordergrund stehen sowohl ein stochastisches als auch ein statisches Fehlermodell. Mit einer stochastischen Verteilung von Bitfehlern lässt sich der Normalbetrieb des LVDS-Systems simulieren. Hier ist es erforderlich, realitätsnahe Parameter zu verwenden. Da noch keine experimentelle Auswertung am realen Kompaktsystem bezüglich der auftretenden Fehleranzahl und -häufigkeit durchgeführt wurde, beruhen diese Parameter auf hypothetischen Annahmen.
Für das stochastische Fehlermodell wird eine Gleichverteilung von Bitfehlern über die gesamte Datenwortbreite angenommen. Die Wahrscheinlichkeiten für die Fehleranzahl in einem Datenwort kann wie folgt berechnet werden:

Ein-Bit-Fehler:

$$\begin{aligned} P(X = 1) &= Datenwortbreite \cdot p_{Bit} \\ &= 32 \cdot p_{Bit} \end{aligned}$$

$p_{Bit} \ldots$ Wahrscheinlichkeit für das Kippen eines Bits

Zwei-Bit-Fehler:

$$\begin{aligned} P(X = 2) &= (Datenwortbreite \cdot p_{Bit}) \cdot ((Datenwortbreite - 1) \cdot p_{Bit}) \\ &= (32 \cdot p_{Bit}) \cdot (31 \cdot p_{Bit}) \end{aligned}$$

Allgemeine Formel:

$$\begin{aligned} P(X = x) &= \prod_{i=0}^{x-1} (Datenwortbreite - i) \cdot p_{Bit} \\ &= \prod_{i=0}^{x-1} (32 - i) \cdot p_{Bit} \end{aligned}$$

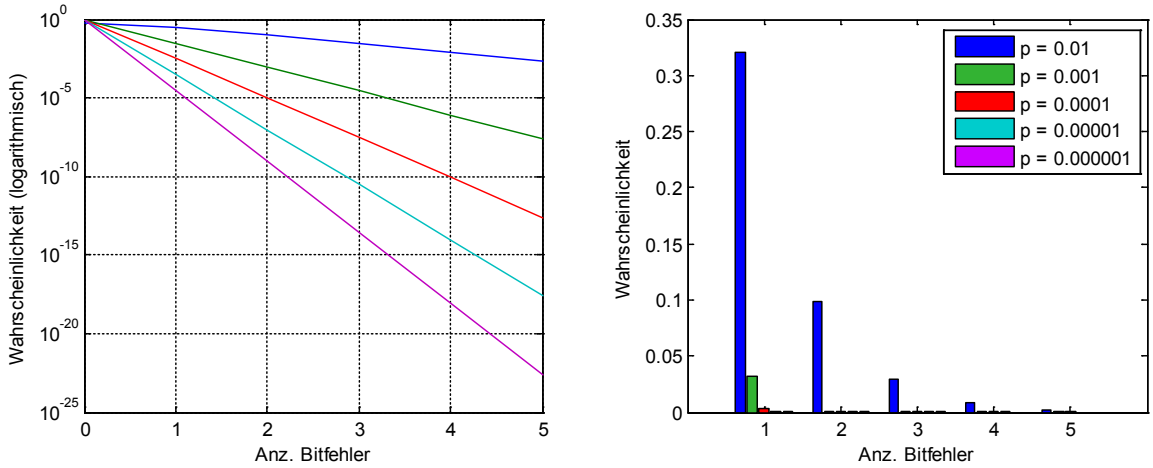



Abb. 5.10.: Bit-Fehler Wahrscheinlichkeiten

Der Verlauf der Wahrscheinlichkeiten für eine unterschiedliche Anzahl von Fehlern wird in Abbildung 5.10 gezeigt.

Um defekte Übertragungsleitungen zu simulieren, wird ein statisches Fehlermodell verwendet. So können die Auswirkungen von 1-, 2- und k-Bit-Fehlern ausgewertet werden.

### 5.2.3. Empfangsauswertung

Die Empfangslogik wertet die empfangenen Pakete des LVDS-Knoten aus. Da der Inhalt der Pakete aus Pseudozufallszahlen besteht, kann dieser leicht nachvollzogen und ausgewertet werden.
Die Ergebnisse der Empfangsauswertung werden protokolliert und können nach der Simulation analysiert werden.

### 5.2.4. Protokollierung

Für eine vollständige Analyse eines Simulationsdurchlaufs wird eine ausreichende Protokollierung der Simulationskomponenten benötigt. Diese soll eine grafische[4] und textuelle Auswertung ermöglichen.

Folgende Daten werden für jeden Frame erfasst:

**Paketgenerator**

- Receiver-ID
- Länge des generierten Datenpakets
- Art des Frames (Header- oder Datenframe)
- Generierte Daten

**Fehlergenerator**

- Header-ID
- Sender-ID
- Receiver-ID
- Art des Frames (Header- oder Datenframe)
- Fehler in Frame gefunden[5]
- Generierte Anzahl von Bit-Fehlern
- Empfangene Daten

[4] Diagramme für generierte Fehler, Fehler in der Empfangsauswertung etc.

[5] Ist nur bei k-Bit-Fehlern ($k > 1$) in Datenframes relevant. 1-Bit-Fehler werden vom LVDS-Knoten sowohl in Header- als auch in Datenframes beseitigt. Treten k-Bit-Fehler in Headerframes auf, so wird dieses Paket nicht an den nächsten LVDS-Knoten weitergeleitet. Somit können nur k-Bit-Fehler in Datenframes vor dem Fehlergenerator auftreten.

**Empfangsauswertung**

- Header-ID
- Sender-ID
- Empfangene Paketlänge
- Fehler von Empfangsauswertung erkannt
- Fehler von LVDS-Knoten erkannt
- Art des Frames (Header- oder Datenframe)
- Position der Bitfehler
- Empfangene Daten

# 6. Implementierung

## 6.1. Anpassung LVDS-Knoten

Für die Implementierung der EMIF-Bridge mussten kleine Änderungen am LVDS-Knoten vorgenommen werden.
Zunächst wurden zwei zusätzliche Ausgangssignale erstellt, die den Füllstatus des Sende- und Empfangspuffers[1] enthalten.
Zusätzlich wurde von Raik Schulze ein Fehler in der Implementierung des LVDS-Knotens behoben. Dieser Fehler führte dazu, dass ein kurzes LVDS-Paket ein langes EMIF-Paket teilweise überschreibt. Geändert wurde lediglich die Statemachine des Ressource-Controllers.

## 6.2. EMIF-Bridge

### 6.2.1. EMIF-Zugriffe

Durch Lese- und Schreibzugriffe der EMIF-Schnittstelle können neben der normalen Paketübertragung im LVDS-Ring weitere Funktionen der EMIF-Bridge angesprochen werden. Schreibzugriffe ermöglichen das Senden von Daten in den LVDS-Ring und den Zugriff auf Kontrollregister der Bridge. Lesezugriffe erlauben das Empfangen von Daten und das Auslesen von Statusregistern.
Um die Zugriffe auf diese Teilfunktionen differenzieren zu können, ist jedem Bereich eine eindeutige Adresse zugewiesen, die bei einem Lese- oder Schreibzugriff auf den

[1] *emif_send_buffer_status* und *emif_receive_buffer_status* (siehe Abbildung 4.1)

Adressbus gelegt werden muss. Tabelle 6.1 enthält die Zuordung der Adressen zu ihren Teilfunktionen.

| Adresse | Lesezugriff | Schreibzugriff |
|---|---|---|
| 0 | Frame eines Pakets vom LVDS-Knoten empfangen | Frame eines Pakets an den LVDS-Knoten senden |
| 1 | Interrupt Statusregister | Interrupt Kontrollregister |
| 2 | Globales Statusregister | Globales Kontrollregister |

Tab. 6.1.: Adresszuordnung

### 6.2.2. Sendelogik

Durch einen Schreibzugriff des EMIF auf Adresse 0 können Datenpakete in den LVDS-Ring übermittelt werden.
Wie bereits in Abschnitt 5.1.1 angesprochen, müssen die Paketframes vollständig zwischengespeichert werden, um eine protokollgerechte Übertragung an den LVDS-Knoten sicherzustellen. Zu diesem Zweck wird ein Dual Port RAM verwendet, um gleichzeitig einen Schreibzugriff der EMIF-Schnittstelle und die Datenübertragung zwischen Bridge und LVDS-Knoten zu ermöglichen. Für die Vermittlung zwischen EMIF-Schnittstelle und RAM-Block ist die Signallogik verantwortlich. Die Vermittlung zwischen RAM-Block und LVDS-Knoten wird anschließend von der Übertragungslogik übernommen.

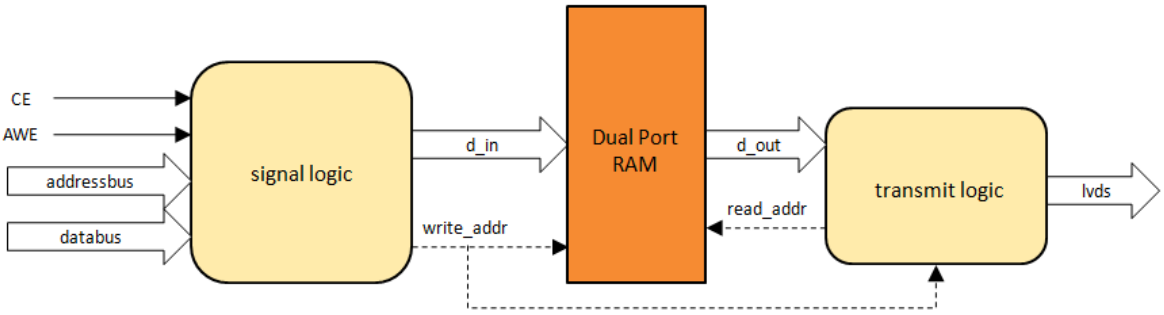



Abb. 6.1.: Aufbau der Sendelogik

In Abbildung 6.1 ist der Aufbau der Sendelogik dargestellt. Dateneingang und Schreibadresse des RAM-Blocks werden mit der Signallogik, Datenausgang und Leseadresse mit der Übertragungslogik verknüpft.

In Abschnitt 2.3.3 wurde der Signalverlauf für einen Schreibzugriff des EMIF erläutert. Dieser wird von der Signallogik (*signal logic*) ausgewertet. Sofern ein Schreibzugriff auf Adresse 0 vorliegt, werden mit Beginn der STROBE-Phase die Daten vom Datenbus an den Dateneingang des Dual Port RAM angelegt. Infolgedessen werden diese Daten auf die aktuelle Schreibadresse (*write_addr*) gespeichert.
Mit dem Ende der STROBE-Phase wird diese Adresse inkrementiert und an den Dateneingang eine 0 angelegt. Dadurch wird der Speicherinhalt der nächsten Speicherzelle überschrieben und verhindert, dass die Übertragungslogik diesen falsch interpretiert.

Die Übertragungslogik übergibt ein Datenpaket an den LVDS-Knoten, sobald dieses vollständig im RAM-Block abgelegt wurde. Abbildung 6.2 zeigt den Ablauf im RAM-Block anhand eines Beispiels mit einem Datenpaket der Länge 2. Zu Beginn zeigen Schreib- und Leseadresse auf dieselbe Speicherzelle, deren Inhalt Null ist (A). Der Inhalt der anderen Speicherzellen ist undefiniert U. Er kann sowohl ein Headerframe als auch ein Datenframe eines zuvor gesendeten Pakets sein.
Der erste EMIF-Schreibzugriff enthält den Headerframe H eines Datenpakets. Dieser wird in die aktuelle Speicherzelle geschrieben, anschließend wird die Schreibadresse inkrementiert. Der Inhalt der nachfolgenden Speicherzelle wird mit dem Ende der STROBE-Phase gelöscht (B).
Anschließend werden die Datenframes D an die EMIF-Bridge gesendet. Diese werden sequentiell im RAM abgelegt (C). Da die Leseadresse (*read_addr*) auf den aktuellen Header zeigt, liegt dieser am Datenausgang des Dual Port RAMs an. Somit ist der Übertragungslogik die Länge des Datenpakets bekannt.
Mit der letzten Inkrementierung der Schreibadresse wird die Bedingung[2]
„$write\_addr \geq read\_addr + length + 1$“ wahr. Das veranlasst die Übertragungslogik, das vollständig eingetroffene Paket an den LVDS-Knoten zu übertragen. Anschließend zeigen Schreib- und Leseadresse auf dieselbe Speicherzelle (D).

[2]length ist die im Headerframe angegebene Datenframe Anzahl

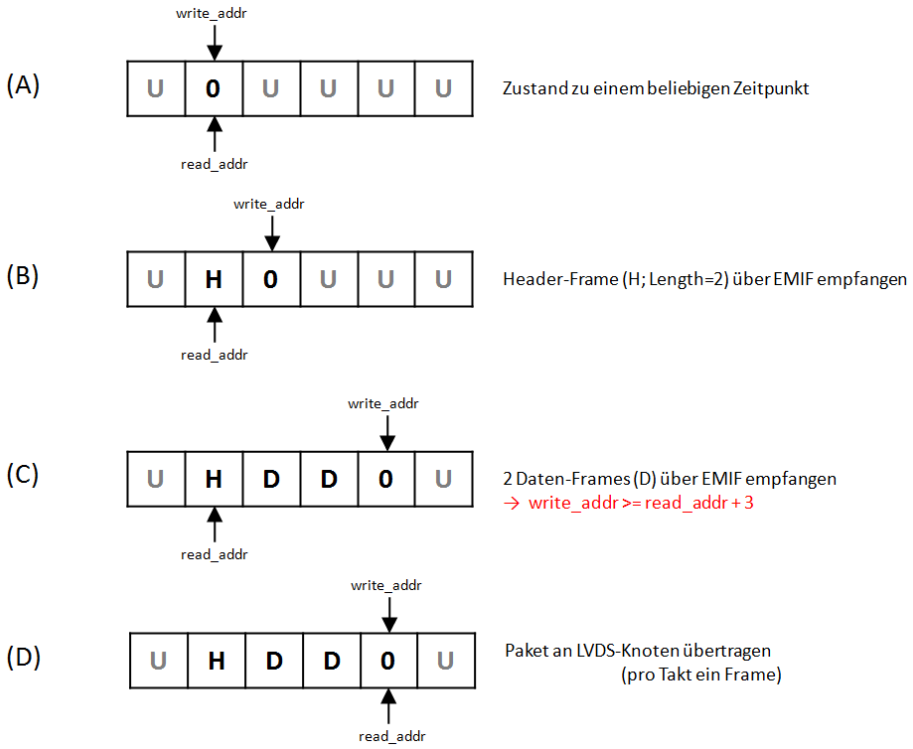



Abb. 6.2.: Ablauf der Zwischenspeicherung

### 6.2.3. Empfangslogik

Die Empfangslogik übernimmt das Auslesen von empfangenen Datenpaketen aus dem LVDS-Knoten und leitet diese bei einem Lesezugriff der EMIF-Schnittstelle (auf Adresse 0) an den DSP weiter.
Vom LVDS-Knoten erhält die Empfangslogik 33 Bit, von denen das höchstwertige ein Fehler-Bit repräsentiert und die restlichen 32 Bit den empfangenen Frame enthalten. Das Fehler-Bit ist 1, wenn der LVDS-Knoten einen Zwei-Bit-Fehler erkannt hat, der bei der Datenübertragung entstanden ist.
Der 32 Bit große Frame enthält 26 Bit relevante Daten für den DSP und 6 Paritätsbits, die vom LVDS-Knoten erzeugt werden. Da die Paritäten für den DSP keine wichtigen Informationen bereitstellen, werden sie von der EMIF-Bridge durch Status-Informationen ersetzt. Der an den DSP zu übermittelnde Frame wird in einem Register (*Message-Register*) zwischengespeichert und bei einem Lesezugriff auf Adresse 0 auf den Datenbus gelegt. Abbildung 6.3 und Tabelle 6.2 zeigt den Aufbau des Registers.

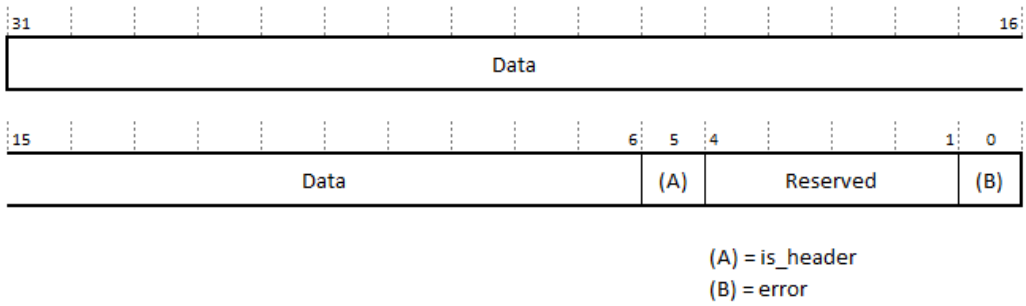



Abb. 6.3.: Aufbau Message-Register

| Bezeichnung | Position | Beschreibung |
|---|---|---|
| Data | 31..6 | Inhalt des Datenframes |
| is_header | 5 | Enthält eine 1, wenn der übertragene Frame ein Header ist; |
| error | 0 | Zwei-Bit-Fehler von LVDS-Knoten erkannt |

Tab. 6.2.: Belegung des Message-Registers

Die Vermittlung zwischen LVDS-Knoten und EMIF-Bridge übernimmt die Frame-Kontrolllogik (siehe Abbildung 6.4).
Um eine genaue Aussage darüber treffen zu können, ob der Empfangspuffer des LVDS-Knoten noch Frames enthält, sind zwei Zähler implementiert. Der Paketzähler (*package_counter*) erfasst die vom LVDS-Knoten erhaltenen Paket-Requests (*emif_sender_request*) und enthält somit die Anzahl der Pakete, die im Empfangspuffer gespeichert sind. Mit jedem *emif_sender_request* wird dieser Zähler inkrementiert. Wird ein Header-Frame, also ein neues Datenpaket, in das Message-Register geladen, so wird der Zähler um 1 verringert.
Der zweite Zähler ist der Paketlängen-Zähler (*length_counter*). Dieser enthält die noch zu ladende Anzahl an Frames des aktuellen Datenpakets. Mit jedem zulässigen Lesezugriff[3] wird dieser Counter dekrementiert. Ist der Zählerwert Null, so wird der Wert des Paketzählers überprüft. Ist dieser ebenfalls Null, so enthält der Empfangspuffer des LVDS-Knotens keine weiteren Datenpakete. Andernfalls wird der Header-Frame des neuen Datenpakets in das Message-Register geladen und der Paketlängen-Zähler

[3]Zulässig bedeutet hier, dass zuvor ein neuer Frame vom LVDS-Knoten bereitgestellt wurde.

auf die Größe des Datenpakets gesetzt.
Informationen über die Verfügbarkeit eines Frames im Message-Register, den Fehler-Status und die Art[4] des abgespeicherten Frames sowie die noch zu ladende Anzahl an Frames im aktuellen Datenpaket werden zusätzlich im Globalen Statusregister (Abschnitt 5.1.4) abgespeichert und stehen dem DSP jederzeit zur Verfügung.

Erfolgt ein EMIF-Lesezugriff ohne das im LVDS-Knoten neue Datenframes vorhanden sind, wird der zuletzt zwischengespeicherte Frame ausgegeben, da das Message-Register nur bei neuen Frames aktualisiert wird.

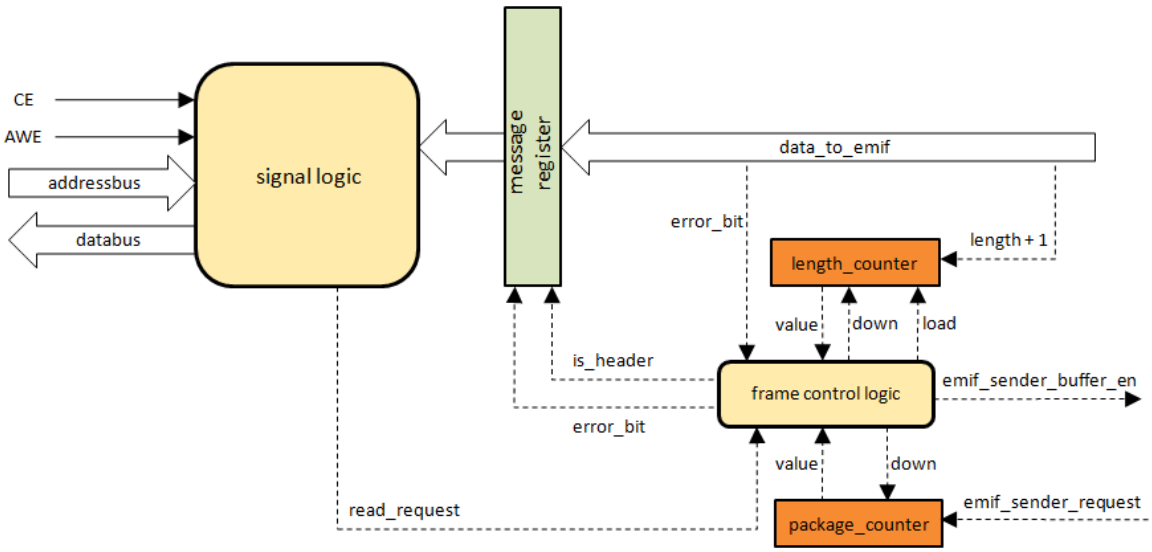



Abb. 6.4.: Aufbau Empfangslogik

## 6.2.4. Interruptlogik

Die Implementierung der Interruptlogik ist an den Aufbau des Programmable Interrupt Controllers (PIC) 8259A [Int88] angelehnt. Der Aufbau ist in Abbildung 6.5 dargestellt. Die Interruptlogik ermöglicht die Behandlung von vier Ereignissen. Tabelle 6.3 zeigt eine Aufstellung der Ereignisse und der Signale, die zu einer Interruptauslösung führen. Der *Interrupt-Generator (IG)* wertet die Signale aus und generiert einen Interrupt-Vektor. Der Interrupt-Vektor besteht aus 4 Bit, von denen jedes ein Interrupt-Ereignis repräsentiert (vgl. irq_event Tabelle 5.2).

[4] Header- oder Datenframe

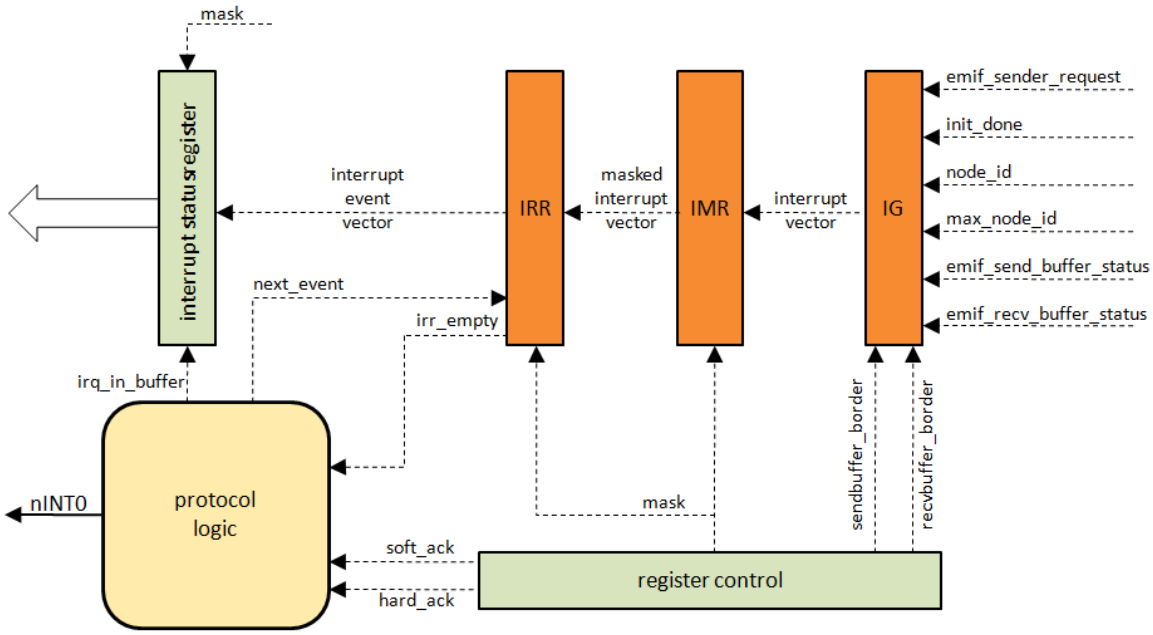



Abb. 6.5.: Aufbau Interruptlogik

Das *Interrupt-Mask-Register (IMR)* kontrolliert, ob die übergebenen Interrupt-Anfragen deaktiviert („maskiert“) sind und leitet den maskierten Interrupt-Vektor an das *Interrupt-Request-Register (IRR)* weiter.
In diesem ist für jedes Interrupt-Ereignis ist ein eigener Zähler vorgesehen. Dieser wird inkrementiert, sobald das jeweilige Ereignis im *masked_interrupt_vector* gesetzt wird. Das Signal *next_irq* löst ein Dekrementieren des Counters aus.
Der Counterwert hat eine Minimalgrenze von 0 und eine Maximalgrenze von 255. Ist diese Maximalgrenze erreicht, gehen die nachfolgenden Interrupt-Ereignisse verloren. Ein Reset des Counterwerts auf 0 wird ausgelöst, wenn die gesamte Interruptlogik zurückgesetzt oder das jeweilige Interrupt-Ereignis deaktiviert wird.

| Ereignis | Signale/Auslösebedingung |
|---|---|
| Paket im LVDS-Knoten empfangen | $emif\_sender\_request = 1$ |
| Kommunikations-Parameter geändert | $init\_done(t) \neq init\_done(t-1) \wedge$<br>$node\_id(t) \neq node\_id(t-1) \wedge$<br>$max\_node\_id(t) \neq max\_node\_id(t-1)$ |
| Grenze in Sendepuffer unterschritten | $emif\_send\_buffer\_status(t) < sendbuffer\_border \vee$<br>$emif\_send\_buffer\_status(t-1) \geq sendbuffer\_border$ |
| Grenze in Empfangspuffer überschritten | $emif\_recv\_buffer\_status(t) > recvbuffer\_border \vee$<br>$emif\_recv\_buffer\_status(t-1) \leq recvbuffer\_border$ |

Tab. 6.3.: Interruptereignisse

Die Protokolllogik (*protocol logic*) behandelt das Kommunikationsprotokoll mit dem DSP. Wenn das IRR Interrupt-Ereignisse enthält, so wird ein Interrupt ausgelöst. Die Protokolllogik erwartet anschließend eine Bestätigung des Prozessors.
Erfolgt ein Soft-Acknowledgment, so wird ein neuer Interrupt-Vektor in das Statusregister geladen[5], der Interruptzustand hingegen nicht verlassen.
Durch ein Hard-Acknowledgment verlässt die Protokolllogik den Interruptzustand, wodurch die IRQ-Leitung (*nINT0*) zurückgesetzt wird. Um dem Prozessor genügend Zeit zu geben, die eigene Interrupt Service Routine zu verlassen, geht die Protokolllogik in den Wartezustand *IRQ_WAIT* über. Erst nach Ablauf einer definierten Zeit (100 $\mu$s) kann ein neuer Interrupt ausgelöst werden.
Abbildung 6.6 zeigt die dafür zuständige Finit State Machine[6] (FSM) der Protokolllogik.

[5] Ausgelöst durch *next_irq*
[6] Der Übersichtlichkeit halber wurde in der Darstellung auf die Eigenschleifen verzichtet. Die FSM verbleibt solang in dem Zustand, bis eine der Kantenbedingung wahr wird.

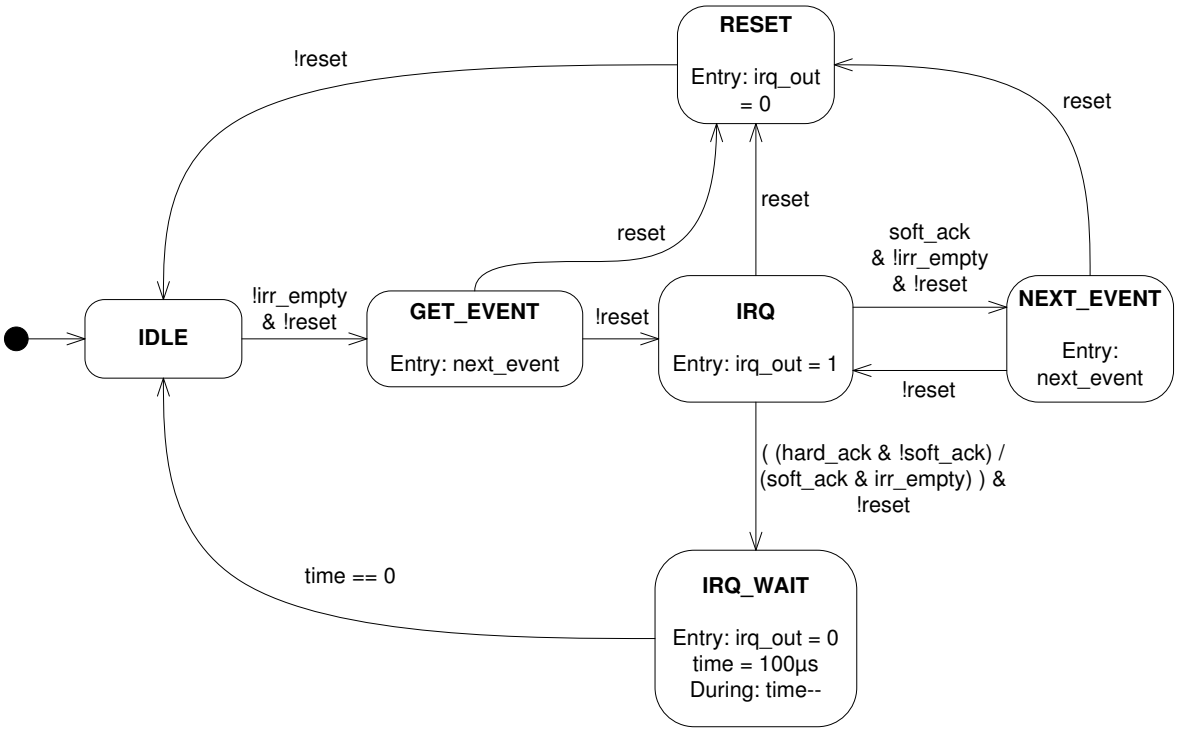



Abb. 6.6.: FSM der Interrupt-Protokolllogik

### 6.2.5. Reset

Die drei Teilkomponenten der EMIF-Bridge können über das Globale Steuerregister (vgl. Abschnitt 5.1.5) einzeln zurückgesetzt werden. Dies hat folgende Auswirkungen:

**Sendelogik** Ein Reset der Sendelogik führt zum Rücksetzen der Lese- und Schreibadresse des Zwischenspeichers auf 0. Anschließend wird diese Speicherzelle zusätzlich gelöscht.

**Empfangslogik** Durch einen Reset der Empfangslogik werden die beiden Zähler (Paket- und Paketlängen-Zähler) zurückgesetzt. Der aktuell abgespeicherte Frame wird gelöscht, so dass das Message-Register keine Daten enthält.

**Interruptlogik** In der Interruptlogik werden durch ein Reset die Ereignis-Zähler auf 0 gesetzt. Zudem geht die Protokolllogik in einen Resetzustand und anschließend in den Idlezustand über.

Ein Reset wird ausgelöst, wenn im Kontrollregister an der entsprechenden Stelle eine 1 erkannt wird. Anschließend ist das Reset-Signal für die Teilkomponenten für 5 Takte aktiv, dies sichert das vollständige Zurücksetzen der Logik.
Eine besondere Relevanz hat dies beim Reset der Sendelogik. Ein Takt wird benötigt, um Lese- und Schreibadresse des Dual-Port-RAMs auf 0 zu setzen. Anschließend wird der Inhalt der Speicherzelle gelöscht. Bis am Datenausgang des RAM-Blocks dieser Inhalt anliegt, vergehen weitere 2 Takte. Dies ergibt unter Beachtung einer zusätzlichen Sicherheitszeit von einem Takt die benötigten 5 Takte für einen vollständigen Reset.

## 6.3. EMIF-Bridge Testumgebung

Um die EMIF-Bridge auf der Zielhardware einzeln überprüfen zu können, wurde eine Testumgebung erstellt (Test_Unit siehe Abbildung C.4), diese ist mit der EMIF-Bridge im Matlab/Simulink-Modell *emif_bridge.mdl* (auf der CD in *\Matlab\EMIF_Bridge*) zu finden. Diese Testumgebung soll einerseits die Schnittstellenfunktionen des LVDS-Knotens nachbilden und andererseits eine leichte Kontrolle der EMIF-Bridge Funktionen ermöglichen.
Die entstandene Test Unit enthält zu diesem Zweck eine FSM (Abbildung 6.7), die die Schnittstelle des LVDS-Knotens zur EMIF-Bridge simuliert, und einen Speicher für die gesendeten Pakete.

Sequentiell ankommende Paketframes werden im Speicher abgelegt. Diese können von der EMIF-Bridge über das Signal *emif_sender_buffer_en* abgerufen werden. Zusätzlich setzt die Testumgebung das Errorbit jedes zweiten Frames, der zurück an die EMIF-Bridge übertragen wird, auf 1. Dadurch kann die Grundfunktionalität – das Senden und Empfangen von Paketen sowie das Globale Statusregister – getestet werden.
Der Füllstand des Speichers wird mit den Status-Eingängen (*emif_send_buffer_status, emif_recv_buffer_status*) der EMIF-Bridge verbunden. Dies ermöglicht das Überprüfen der Interruptfunktionalität.

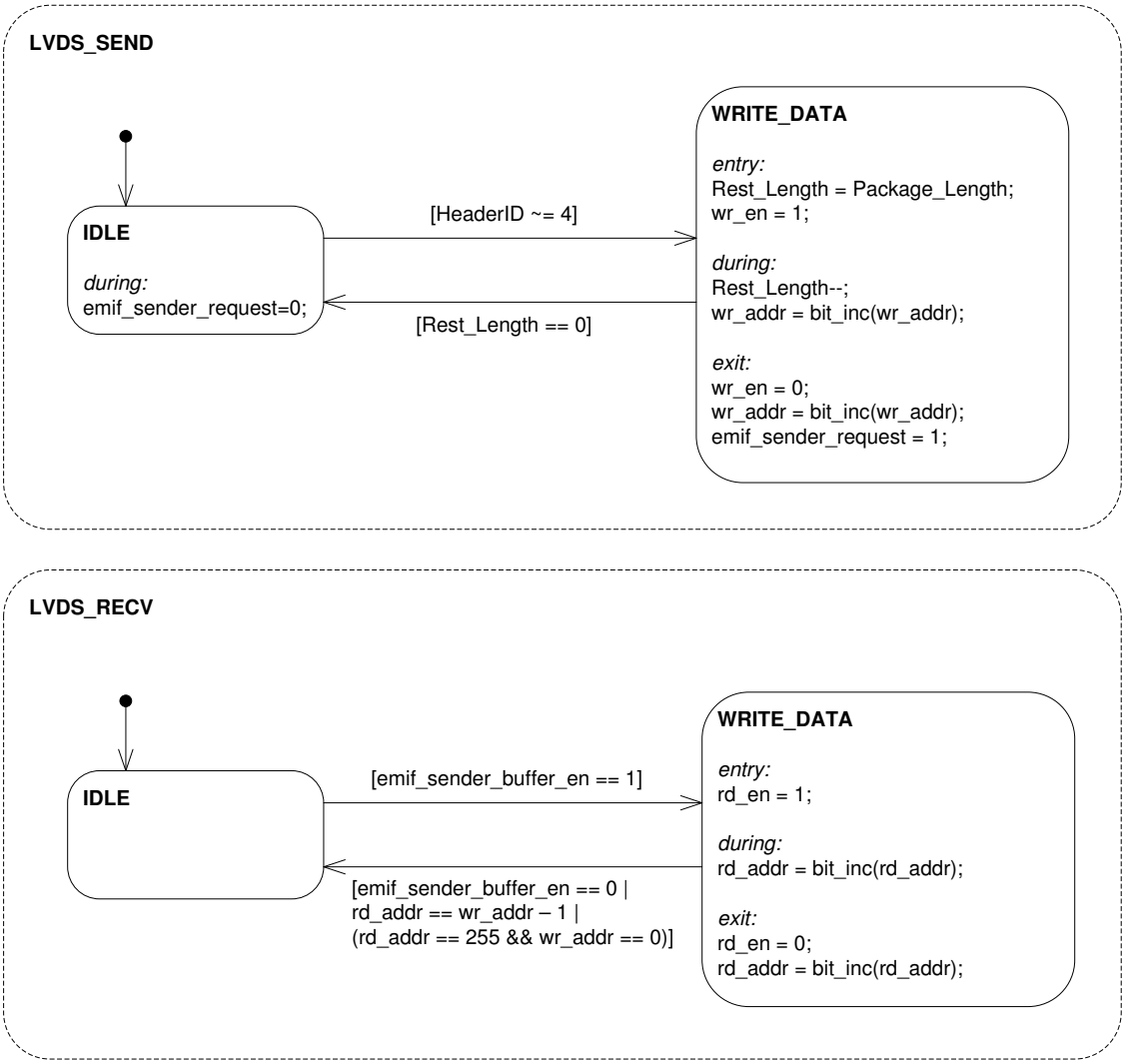



Abb. 6.7.: FSM der Testumgebung (in Test_Unit von Abbildung C.4)

# 7. User Guide

Im Nachfolgenden wird die Nutzung der EMIF-Bridge aus Sicht der DSP-Software und die Nutzung der Simulationsumgebung in Matlab/Simulink für weitere Tests erläutert.

## 7.1. EMIF-Bridge

### 7.1.1. Interrupt Service Routine

Durch das Handshake-Verfahren mit der EMIF-Bridge ist der DSP an die Vorgaben des Interrupt-Kommunikationsprotokolls gebunden. Dabei sollte vor allem auf den richtigen Gebrauch der zwei Bestätigungstypen geachtet werden.
Nach einem Hard Acknowlegement muss die ISR zeitnah verlassen werden. Die EMIF-Bridge geht nach dem Rücksetzen der Interruptleitung für 100 $\mu$s in einen Wartezustand über. Erst anschließend kann ein neuer Interrupt von der EMIF-Bridge ausgelöst werden.
Ein Soft Acknowledgement darf nicht an die EMIF-Bridge gesendet werden, wenn das Interrupt Request Register leer ist. Die EMIF-Bridge behandelt in so einem Fall das Soft Acknowledgement zwar wie ein Hard Acknowledgement, es ist jedoch nicht sichergestellt, dass der DSP auch die Interrupt Service Routine verlässt.
Abbildung 7.1 zeigt den möglichen Ablauf in der ISR.

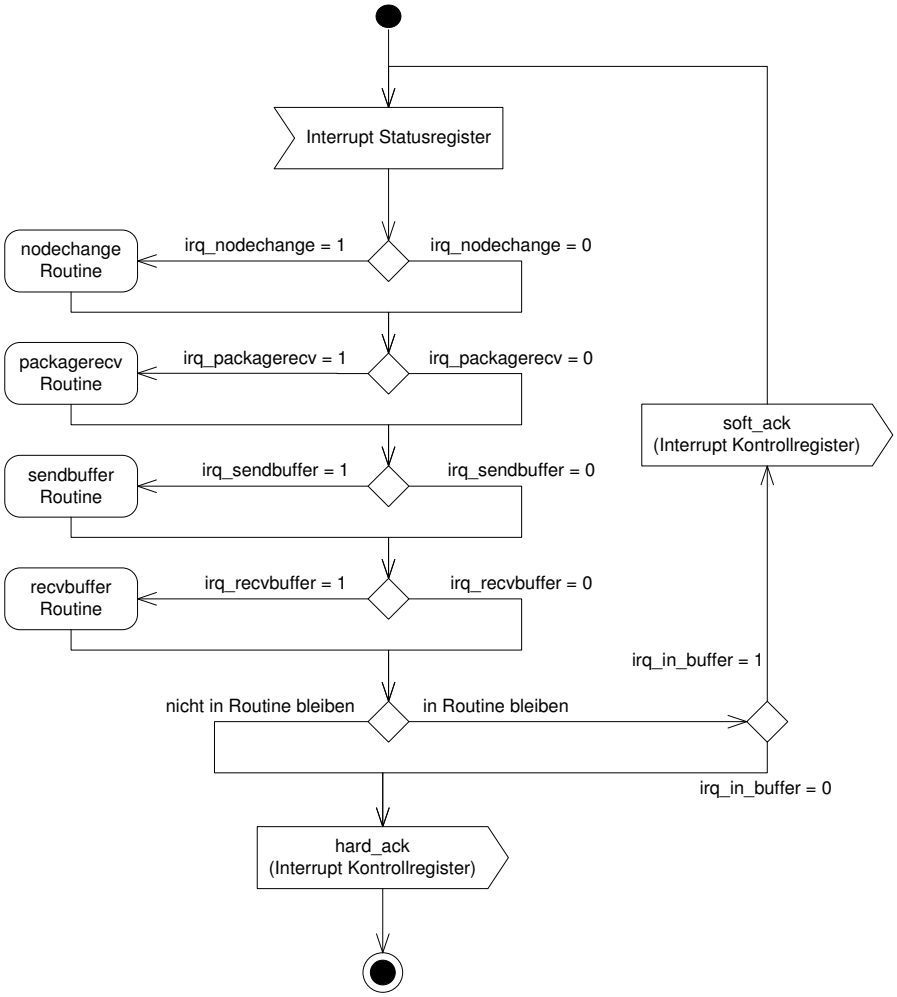



Abb. 7.1.: Möglicher Ablauf ISR

## 7.1.2. Bedeutung und Anwendung der Resets

### Reset Sendelogik

Das Reset der Sendelogik bewirkt das Rücksetzen von Schreib- und Leseadresse auf den internen Zwischenspeicher. Aus diesem Grund muss sichergestellt sein, dass ein Reset nicht während der Übertragung eines Datenpakets von der EMIF-Bridge an den LVDS-Knoten ausgelöst wird. Die Übertragung benötigt soviel Takte, wie Frames im Paket vorhanden sind. Erst anschließend sollte ein Reset verwendet werden.

Das Rücksetzen der Sendelogik ist bei jedem Start des DSPs angebracht. Sollte der Start durch einen Reset ausgelöst worden sein, kann keine klare Aussage darüber getroffen werden, ob sich ein unvollständiges Datenpaket im Zwischenpuffer befindet. Dieses wird durch ein Rücksetzen der Sendelogik entfernt.

### Reset Empfangslogik

Ein Reset der Empfangslogik bewirkt das Leeren des Message-Registers und das Rücksetzen von Paket- und Paketlängenzähler auf 0. Da der LVDS-Knoten noch keine Möglichkeit bietet, den internen Empfangspuffer zu löschen, sollte ein Reset der Empfangslogik nur in Ausnahmefällen eingesetzt werden. Enthält dieser Puffer noch Daten, so kann eine korrekte Funktionalität der Empfangslogik nach dem Rücksetzen nicht gewährleistet werden.

Ein Reset der Empfangslogik ist bei fatalen Fehlern im LVDS-Ring ratsam. Hervorgerufen durch n-Bit-Fehler ($n \geq 2$) im Headerframe, kann das gesamte Datenpaket beschädigt werden[1] und eine Zuordnung des Pakets zu einem Empfänger nicht mehr möglich sein. Da nachfolgende Pakete davon ebenfalls betroffen sind, ist ein Rücksetzen sowohl der Empfangslogik, als auch des LVDS-Knotens notwendig.

### Reset Interruptlogik

Ein Reset der Interruptlogik bewirkt das Löschen aller anstehenden Interrupt-Ereignisse und das Rücksetzen der Protokolllogik.

Eine Nutzung bietet sich bei jedem fatalen Fehler im LVDS-Ring an.
Bei einem Rücksetzen von Sende- oder Empfangslogik sollten die jeweiligen Ereignisse für ein Unterschreiten der Sendepuffergrenze bzw. ein Überschreiten der Empfangspuffergrenze aus dem Interrupt Request Register gelöscht werden. Dies ist durch einmaliges Deaktivieren und anschließendes Aktivieren des Interrupt-Ereignisses möglich.

[1] Bitfehler im Length-Parameter

Es sollte darauf verzichtet werden, einen Reset der Interruptlogik in der ISR auszulösen. Dies kann dazu führen, dass die EMIF-Bridge eine neue Unterbrechung erzeugt, bevor der DSP die Routine verlassen hat. Die EMIF-Bridge würde im Interrupt-Zustand verbleiben, ohne dass eine geeignete Behandlung des Prozessors ausgelöst wurde.
Ist ein Reset trotzdem nötig, sollten zuvor alle Ereignisse in der Interrupt-Maske deaktiviert werden. Dies verhindert ein ungewünschtes Auslösen des Interrupts.

## 7.2. Simulationsumgebung

### 7.2.1. Testlauf erstellen

Die Steuerung der Simulationskomponenten Paketgenerator und Fehlergenerator übernimmt zur Simulationszeit die Testlogik (*Test_logic*). Über eine Maske der Testlogik können grundlegende Einstellungen für den Testlauf vorgenommen werden:

| | |
|---|---|
| **Start Delay:** | Verzögerung bis der Paketgenerator gestartet wird (nötig um die Initialisierung des LVDS-Rings abzuschließen) |
| **Run Time:** | Dauer des Testlaufs |
| **Test Nr:** | Angabe der Testnummer |
| **only Error after Node1:** | 1 – nur Fehlergenerator nach Knoten 1 ist aktiv<br>0 – alle Fehlergeneratoren sind aktiv |
| **Delay 1:** | Verzögerung zwischen der Aktivierung des Paketgenerators und der Aktivierung des Fehlergenerators |
| **Delay 2:** | Verzögerung zwischen der Deaktivierung des Fehlergenerators und der Deaktivierung des Paketgenerators |

Zu beachten ist, dass $RunTime \geq Delay1 + Delay2$.

Simulationseinstellungen für den Paketgenerator und den Fehlergenerator werden zu einem Testfall zusammengefasst (Auswahl durch [*Test Nr*]).

Testfälle werden im *Test_logic*-Block als Subsystem definiert. Folgende Einstellungen sind möglich:

### Paketgenerator

| | |
|---|---|
| **Enable:** | Aktiviert den Paketgenerator im Testlauf |
| **Reset:** | Zurücksetzen des Paketgenerator |
| **min. Packagelength:** | minimale Paketlänge (mindestens 1) |
| **max. Packagelength:** | maximale Paketlänge |
| **min. Datafield:** | minimaler Wert des *Data*-Parameters im Paketheader (nötig für die Pseudozufallsdaten in den Datenframes) |
| **max. Datafield:** | maximaler Wert des *Data*-Parameters |
| **min. Idle-Frames:** | minimale Pause zwischen zwei generierten Datenpaketen |
| **max. Idle-Frames:** | maximale Pause |

### Fehlergenerator

| | |
|---|---|
| **Enable:** | Aktiviert den Fehlergenerator im Testlauf; |
| **Enable non-Datapackages:** | 1 – Fehler werden auch außerhalb von Datenpaketen generiert;<br>0 – Fehler werden nur in Datenpaketen generiert; |
| **Error Probability:** | Fehlerwahrscheinlichkeit eines Bitfehlers bezogen auf einen Teiler von 1.000.000.000;<br>Beispiel: Wenn der Parameter *Error Probability* auf 1.000.000 gesetzt wird, ist die Fehlerwahrscheinlichkeit in der Simulation $1.000.000/1.000.000.000 = 0,001 = 0,1\%$; |
| **max. Errors:** | Maximale Anzahl an Fehlern pro Frame; |
| **Trivial Errors:** | Triviale Fehler:<br>0 – keine Trivialfehler;<br>1 – ausgehender Frame besteht nur aus Nullen;<br>2 – ausgehender Frame besteht nur aus Einsen; |
| **Static Error-mask:** | Statische Fehlermaske, die zusätzlich über die generierten Fehler gelegt wird; |
| **Start Option:** | 0 – Fehlergenerator startet sofort;<br>1 – Fehlergenerator startet bei erstem eingehenden Headerframe;<br>2 – Fehlergenerator startet bei erstem eingehenden Datenframe; |
| **Header-ID Option:** | 0 – Fehler werden in Header- und Datenframe erzeugt;<br>1 – Fehler nur in Headerframe;<br>2 – Fehler nur in Datenframe; |

Durch die statische Fehlermaske können feststehende Fehler definiert werden. Beispielsweise erzeugt eine Fehlermaske mit dem Wert 17 (10001b) statische Fehler an Bitposition 1 und 5.

### 7.2.2. Protokollierung

Die Simulationskomponenten protokollieren ihr Verhalten zu jedem Simulationsschritt. Dies ist einerseits für die grafische Auswertung nötig, andererseits kann damit der Simulationsverlauf zu jedem Zeitpunkt nachvollzogen werden.
Die Protokolle werden im Matlab-Workspace abgelegt. Jede Zeile enthält einen Simulationsschritt mit folgenden Daten:

| Spalte | Inhalt |
|---|---|
| 1 | Empfänger-ID |
| 2 | Anzahl Datenframes |
| 3 | 1 wenn es ein Headerframe ist, 0 sonst |
| 4 | Generierte Frames |

Tab. 7.1.: Protokoll Paketgenerator

| Spalte | Inhalt |
|---|---|
| 1 | Header-ID |
| 2 | Sender-ID |
| 3 | Anzahl der Datenframes |
| 4 | 1 wenn es ein Headerframe ist, 0 sonst |
| 5 | Fehler in Frame von Empfangsauswertung erkannt |
| 6 | Fehler in Frame von LVDS-Knoten erkannt (*error_bit*) |
| 7 | empfangener Frame |

Tab. 7.2.: Protokoll Empfangsauswertung

| Spalte | Inhalt |
|---|---|
| 1 | Header-ID |
| 2 | Sender-ID |
| 3 | Empfänger-ID |
| 4 | 1 wenn es ein Headerframe ist, 0 sonst |
| 5 | Fehler in Frame von Fehlergenerator erkannt |
| 6 | Anzahl generierter Fehler |
| 7 | Fehlermuster (enthält die Positionen der generierten Fehler) |
| 8 | empfangener Frame |

Tab. 7.3.: Protokoll Fehlergenerator

Eine grafische Auswertung kann mittels des M-Files *simulation_results.m* vorgenommen werden. Die Ausgabe erfolgt in Balkendiagrammen:

**Figure 1 - Gesendete Pakete:** Diese Auswertung zeigt für jeden Knoten ein eigenes Balkendiagramm,welches die Anzahl der generierten Pakete getrennt nach den Empfängern enthält.

**Figure 2 - Empfangene Pakete:** Für jeden Knoten wird ein Balkendiagramm erstellt, das die Anzahl der empfangenen Pakete und erkannten Fehler getrennt nach der Sender-ID darstellt.

**Figure 3 - Generierte Fehler:** Diese Auswertung enthält Balkendiagramme für jeden Fehlergenerator und zeigt eine Übersicht über die generierten Fehler. Dabei ist zu beachten, dass ein 2-Bit-Fehler unterschiedliche Bedeutungen haben kann. Entweder wurden zwei fehlerhafte Bits generiert oder ein Fehler im Datenframe wurde erkannt, d.h. der Fehlergenerator auf der davor liegenden Übertragungsleitung hat mindestens zwei Bit-Fehler erzeugt.

# 8. Integration und Test

## 8.1. Ergebnisse Dual-Port-RAM Test

Im Rahmen der Bachelorarbeit von Raik Schulze [Sch10] ist eine Testimplementierung entstanden, die das Verhalten des Dual-Port-RAM aus der HDL Coder Bibliothek überprüft. Durch diesen Test soll sichergestellt werden, dass die VHDL-Synthese die gewünschte Funktionalität bereitstellt. Der Test läuft in drei Phasen ab:

1. Zu Beginn erfolgt die Überprüfung des einzelnen Schreib- und Lesezugriffs. Der RAM-Block wird zunächst vollständig mit zufälligen Daten befüllt, anschließend werden alle Speicherzellen ausgelesen und verglichen.
2. In der zweiten Phase wird der RAM mit Nullen beschrieben und somit zurückgesetzt.
3. Die dritte Phase dient der Überprüfung des gleichzeitigen Schreib- und Lesezugriffs. Die RAM-Zellen werden nacheinander beschrieben. Mit einer Verzögerung von zwei Takten wird der Inhalt der Zellen ausgelesen und verglichen.

Für die Statusausgabe des Tests werden die boardeigenen LEDs genutzt, Tabelle 8.1 zeigt die Belegung.

Wird ein Fehler während des Tests erkannt, so wird die Abarbeitung unterbrochen und LED 7 aktiv gesetzt. Den anderen LEDs kann die Test- und Sub-Phase entnommen werden, in denen der Fehler aufgetreten ist.

| LED | Funktion |
|---|---|
| LED 0-2 | Ausgabe der aktuellen Sub-Phase des Tests |
| LED 3-5 | Ausgabe der aktuellen Test-Phase |
| LED 7 | Fehlererkennung, aktiv falls ein Fehler erkannt wurde. |

Tab. 8.1.: LED-Belegung RAM-Test

**Ergebnis:**

Der Test ist ohne Fehler abgelaufen. Die Status-LEDs zeigten Phase 3, Sub-Phase 2 und keinen Fehler an, d.h. der Endzustand wurde ohne Unterbrechung erreicht. Damit ist die Funktionalität der VHDL-Synthese nachgewiesen und der Dual-Port-RAM der HDL Coder Bibliothek kann für weitere Implementierungen genutzt werden.

## 8.2. Integration und Test der EMIF-Bridge

### 8.2.1. Codegenerierung mit dem HDL Coder

Durch das zusätzliche Plugin *HDL Coder* kann das Simulink-Modell in eine synthetisierbare Hardwarebeschreibungssprache überführt werden.
Zunächst muss der HDL Coder entsprechend konfiguriert werden. Dies ist über das HDL Coder Menü (`Tools >> HDL Coder >> Options...`) möglich.
Im HDL Coder Menü kann zunächst das zu übersetzende Subsystem, die Beschreibungssprache und ein Speicherort für die generierten Dateien gewählt werden (siehe Abbildung 8.1).

Im Menü *Global Settings* (siehe Abbildung 8.2) können Einstellungen für den Reset- und Clock-Eingang vorgenommen sowie Prefixe und Postfixe für bestimmte Signalleitungen festgelegt werden. Für diese Optionen können problemlos die defaultmäßig gesetzten Werte genutzt werden.
Der Button *Generate* startet schließlich die Übersetzung.

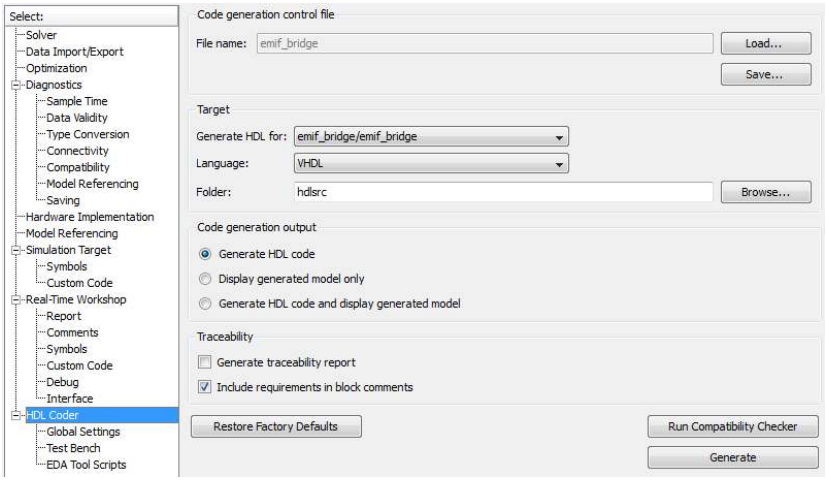



Abb. 8.1.: HDL Coder Optionen

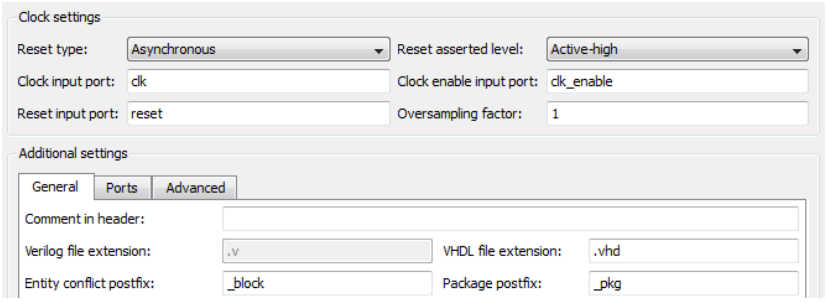



Abb. 8.2.: Globale Einstellungen für den HDL Coder

Eine weitere nützliche Funktionalität des HDL Coders ist die automatische Test Bench Generierung. Dazu muss das zu testende System in eine Testbench eingebettet werden. Die Einbettung der EMIF-Bridge ist in Abbildung C.2 dargestellt.
Der *EMIF-Signalgenerator* erstellt die EMIF-Signale anhand einer Variablen im Matlab Workspace.
Durch den Button *Generate Test Bench* im Menü Test Bench werden alle benötigten Dateien für eine Simulation in ModelSim erstellt. Dies beinhaltet eine VHDL-Datei, die sowohl die Eingangssignale für die Stimulation des Systems, als auch die Ausgangssignale aus der Matlab-Simulation enthält.

### 8.2.2. Verifikation mit ModelSim

Die Verifikation mit ModelSim soll die generierten VHDL-Dateien zusätzlich überprüfen. Da es bereits vorgekommen ist, dass Teile eines Modells (vgl. [Sch10, Abschnitt 7.2]) nicht korrekt übersetzt wurden, soll mit einem ModelSim Test die Vollständigkeit des Systems sichergestellt werden. Dabei wird von der Funktion des HDL Coders, eine Test Bench aus den Matlab Simulationsdaten zu erzeugen, Gebrauch gemacht. Zusätzlich zu den VHDL-Dateien für System und Test Bench erstellt der HDL Coder zu diesem Zweck Scripte für die Ausführung der Simulation im ModelSim.

Nachdem ein neues ModelSim Projekt angelegt wurde, müssen die generierten Dateien in den Projektordner kopiert werden. Anschließend können diese Dateien über das Menü (`Project >> Add to Project >> Existing File...`) in das Projekt eingebunden werden.

Die Script-Dateien dienen der vollständigen Kompilierung und dem Start der Simulation. Sie können über das Kontextmenü oder die ModelSim Kommandozeile ausgeführt werden.
Durch die folgenden Befehle werden die VHDL-Dateien kompiliert:

```
do emif_bridge_compile.do
do emif_bridge_tb_compile.do
```

Der Start der Simulation erfolgt durch:

```
do emif_bridge_tb_sim.do
```

Zu jedem Ausgangssignal des getesteten Systems wird ein Referenzsignal erzeugt, das die Daten der Matlab Simulation enthält. Somit kann leicht nachvollzogen werden, ob das System korrekt übersetzt wurde. Abbildung C.3 zeigt einen Teil des Simulationsergebnisses.

### 8.2.3. Synthese mit Quartus II

Die EMIF-Bridge sowie die Testumgebung werden mit Quartus II synthetisiert. Das Quartus Projekt beinhaltet zu diesem Zweck neben den generierten VHDL-Dateien ein

Top-Level-Design.
Um die Kopplung der Ein- und Ausgänge der EMIF-Bridge an entsprechende FPGA Pins übersichtlich zu gestalten, wird als Top-Level-Design ein Schematic verwendet. Zunächst wird aus der obersten Modellebene der generierten VHDL-Dateien ein Symbol erzeugt. Dieses kann anschließend in das Schematic eingefügt und mit entsprechenden Input- und Output-Pins verbunden werden.
Zu beachten ist, dass der Datenbus des EMIF ein bidirektionaler Bus ist. Um die Bidirektionalität zu gewährleisten, wird die Quartus Megafunction für einen Tri-State Buffer verwendet. In Abhängigkeit von den Steuersignalen des EMIF kann somit der Datenbus getrieben oder vom Datenbus gelesen werden.
Abbildung C.4 zeigt das Top-Level-Schematic.

### 8.2.4. Integrationstest

Die Anbindung der EMIF-Bridge an eine Simulationsumgebung ermöglicht das Überprüfen der Grundfunktionen ohne den Aufbau des vollständigen Kommunikationssystems.
Der Test ist in einem Programm für den DSP realisiert[1] und umfasst vier Phasen:

1. In der ersten Phase werden die Grundfunktionen Senden, Empfangen von Paketen und der Inhalt des globalen Statusregisters überprüft.
   Zu diesem Zweck werden zunächst 10 Datenpakete mittels des EMIFs an den FPGA übergeben. Der Inhalt des Datenframes entspricht einer fortlaufenden Nummer, die bei jedem Frame hochgezählt wird.
   Anschließend erfolgt abwechselnd das Überprüfen des globalen Statusregisters und das Auslesen eines Paketframes. Wird ein Fehler im Statusregister oder im Frame erkannt, so wird der Testdurchlauf abgebrochen.
   Die Interruptgenerierung der EMIF-Bridge ist in dieser Testphase deaktiviert.

2. Die zweite Phase kontrolliert das Verhalten der Interruptlogik bei einem Hard-Acknowledgement. Zunächst werden zwei Datenpakete mit jeweils 20 Datenframes an die EMIF-Bridge gesendet, die anschließend wieder ausgele-

[1] auf der CD in Ordner *\CCStudio\EMIF_Bridge_TEST*

sen werden. Dies erzeugt Interrupts für die Ereignisse „Paket empfangen“, „Empfangspuffergrenze überschritten“, „Sendepuffergrenze unterschritten“.

Die Interrupt Service Routine enthält einen Zähler für jedes Interruptereignis. Um den Test erfolgreich zu beenden, müssen die Zähler folgende Werte enthalten:

- Paket-Empfangen-Zähler = 2
- Knoten-Änderungs-Zähler = 0
- Empfangspuffer-Überschritten-Zähler = 2
- Sendepuffer-Unterschritten-Zähler = 2

3. Die dritte Phase überprüft anschließend das Verhalten der Interruptlogik bei einem Soft-Acknowledgement unter Beachtung des Interrupt-Statusregisters. Zunächst wird durch das Senden eines Datenpakets die Interruptauslösung angeregt.
   In der Interrupt Service Routine wird anschließend das Interrupt-Statusregister überprüft. Ist der Inhalt korrekt, so wird ein neues Datenpaket verschickt, ohne die ISR zu verlassen. Dies veranlasst die EMIF-Bridge, weitere Interruptereignisse zu erzeugen. Anschließend wird nach einem Soft-Acknowledgement das Statusregister erneut überprüft und die Testphase verlassen.

4. In der vierten Testphase werden abschließend Sendelogik, Empfangslogik und Interruptlogik einzeln zurückgesetzt und die Auswirkungen anhand der Statusregister überprüft.

### 8.2.5. Ergebnisse Integrationstest

Aufgrund eines ungeklärten Problems mit dem Quartus II Compiler konnte die EMIF-Bridge, in Verbindung mit der Testumgebung, nicht auf der Zielhardware überprüft werden.
Die Simulation von Quartus II zeigt das gewünschte Verhalten der EMIF-Bridge. Da die Ergebnisse des Quartus Simulators vom Verhalten der realen Hardware abweichen, liegt die Ursache für eine falsche Übersetzung des VHDL-Codes im Compiler.
Die Simulation erfolgt auf Register Transfer Ebene, d.h. die RTL-Synthese verläuft

fehlerfrei, somit werden vermutlich das Technology Mapping bzw. die Optimierungsverfahren des Compilers nicht fehlerfrei ausgeführt. Eine Fehlermeldung des Compilers erfolgt allerdings nicht.
Die durchgeführte Quartus-Simulation *simpleTestbench.vwf* ist auf der CD in Ordner *\Quartus\EMIF_Bridge_TEST* zu finden. In dieser Simulation wird ein Datenpaket mit der Länge 4 an die EMIF-Bridge gesendet. Anschließend wird es durch Lesezugriffe wieder ausgelesen. Der Test mit der Hardware ist in einem Programm für den DSP realisiert (*\CCStudio\EMIF_Bridge_Test\main.c*). In der main-Funktion kann zwischen zwei Tests gewählt werden, durch die Funktion *smallTest()* wird die zu der Simulation äquivalente Verhaltensüberprüfung ausgeführt. Durch Nutzung der Debug-Funktionen von CCStudio ist leicht ersichtlich, dass Simulation und Test zu unterschiedlichen Ergebnissen führen und dass die Synthese von Quartus fehlerhaft verlief.

Das Problem konnte auf die Empfangslogik der EMIF-Bridge eingegrenzt werden. Da die EMIF-Bridge den Headerframe eines neuen Datenpaketes korrekt abruft und dem DSP zur Verfügung stellt, anschließend aber die Datenframes nicht aus der Testumgebung ausliest, wurde vom Compiler die Logik für das *emif_sender_buffer_en*-Signal[2] nicht korrekt übersetzt.

Die Funktionalität der EMIF-Bridge und der Testumgebung konnte somit nur durch geeignete Matlab/Simulink und Quartus II Simulationen überprüft werden.

## 8.3. Test und Ergebnisse Matlab/Simulink Simulation

Für den funktionalen Test des LVDS-Kommunikationssystems, bestehend aus 3 LVDS-Knoten, wurden folgende Testszenarien erstellt und mittels Matlab simuliert:

[2] zu finden in Matlab/Simulink-Modell *\Matlab\EMIF_Bridge\emif_bridge.mdl* in Block *emif_bridge/EMIF_Bridge_with_Testunit/EMIF-Bridge/frame_control_logic*

1. stochastische Fehler

| | | |
|---|---|---|
| 1.1. | stochastische Fehler | Nach Abschluss der Initialisierung werden Übertragungsfehler generiert. Es sind sowohl Datenpaket-Frames als auch Idle-Frames betroffen. Die Bit-Fehler-Wahrscheinlichkeit in diesem Szenarium beträgt $0,1\%$. |

2. Trivialfehler

| | | |
|---|---|---|
| 2.1. | Null | In diesen Testfällen werden nach der Initialisierung für einen bestimmten Zeitraum nur Nullen bzw. Einsen durch den Fehlergenerator übertragen. |
| 2.2. | Eins | |

3. Statische Bit-Fehler (BF)

| | | |
|---|---|---|
| 3.1. | 1-BF (Datenbereich) | Nach der Initialisierung werden in diesen Testfällen 1-Bit-Fehler im Datenbereich (Bits 31..6) bzw. Paritätsbereich (Bits 5..0) jedes Frames erzeugt. |
| 3.2. | 1-BF (Paritätsbereich) | |
| 3.3. | 2-BF (Datenbereich) | Äquivalent zu 3.1. und 3.2. werden in diesen Testfällen 2-Bit-Fehler im Datenbereich und Paritätsbereich erzeugt. Zusätzlich wird im Testfall *gemischt* ein 1-Bit-Fehler in jedem Bereich erstellt. |
| 3.4. | 2-BF (Paritätsbereich) | |
| 3.5. | 2-BF (gemischt) | |

4. Bit-Bouncing

| | | |
|---|---|---|
| 4.1. | einzelne Leitungen | Mit diesen Testfällen wird das Prellen bei der realen Datenübertragung simuliert. Zwischen LVDS-Transmitter und LVDS-Receiver werden Datenworte mit einer Breite von 8 Bit übertragen (Serialisierungsfaktor = 4). Demzufolge enthält der erste Testfall (4.1.) eine statische Fehlermaske, die einen Bitfehler in jedem achten Bit eines Frames erzeugt. Im zweiten Testfall (4.2.) werden in einem zusammenhängenden 8er Block Bitfehler erzeugt. |
| 4.2. | Block | |

Das Matlab/Simulink-Modell mit der Simulationsumgebung ist auf der CD in Ordner *\Matlab\Simulation LVDS\lvds_communication.mdl* zu finden.

### 8.3.1. Ergebnisse

Die grafische Auswertung des Tests ermöglicht in einem ersten Schritt das Erkennen von Fehlern während der Simulation. Zunächst werden die Balkendiagramme[3] für den Paketgenerator und die Empfangsauswertung verglichen. Treten Unterschiede bei den gesendeten und empfangenen Paketen/Frames auf, können die Übertragungsfehler im Balkendiagramm des Fehlergenerators nachvollzogen werden. Der Verlust von Paketen tritt auf, wenn mehr als ein Bit-Fehler im Headerframe eines Pakets erzeugt wurde. Dieses Paket wird vom LVDS-Knoten nicht weitergeleitet und somit von der Empfangsauswertung nicht erfasst.

In einem zweiten Schritt werden anschließend die Simulationsfehler ausgewertet, deren Ursache nicht mit dem Fehlergenerator nachvollziehbar ist. Den Protokollen der Simulationskomponenten kann zu diesem Zweck der genaue Simulationsverlauf entnommen werden.

Die Protokolle und die grafische Auswertung der zuvor beschriebenen Testszenarien sind auf der CD zu finden.

[3] Matlab-Figures auf CD in Ordner *\Matlab\Simulation LVDS\Ergebnisse*

Die Untersuchung der Fehlerursachen ergab folgende Implementierungsfehler in dem von Raik Schulze[Sch10] übernommenen LVDS-Knoten:

1. Test 1.1. ergab einen Implementierungsfehler bei der Fehlerkorrektur im Headerframe eines Datenpakets. Ein 1-Bit-Fehler an Bitposition 28 führt dazu, dass anstatt der Header-ID 5, die Header-ID 4 übertragen wird. Dieser Fehler wird vom LVDS-Knoten nicht korrigiert und der Headerframe wird als Idleframe behandelt. Der Implementierungsfehler konnte mit Test 3.1. bestätigt werden.

2. Test 2.1. und 2.2. ergaben, dass Trivialfehler in Datenframes nicht erkannt werden. Das error_bit wird beim Auslesen des Pakets nicht gesetzt.

# 9. Zusammenfassung und Ausblick

Ziel dieser Arbeit war die modellbasierte Entwicklung einer Schnittstelle, die eine DSP-Ankopplung an ein bestehendes LVDS-Kommunikationssystem ermöglicht. Diese sogenannte EMIF-Bridge umfasst vier grundlegende Funktionalitäten – das Senden von Datenpaketen an einen beliebigen DSP im LVDS-Ring, das Auslesen von empfangenen Datenpaketen zu einem beliebigen Zeitpunkt, das Generieren eines Interrupts bei bestimmten Ereignissen sowie das Bereitstellen von Status- und Steuerregistern.
Zudem wurde eine Simulationsumgebung in Matlab/Simulink erstellt, die einen umfangreichen Test des LVDS-Kommunikationssystems mit zwei oder mehr LVDS-Knoten ermöglicht. Durch eine Parametrisierung der Simulations-Blöcke können zudem unterschiedliche Lastprofile für die Paket- und Fehlergenerierung eingestellt werden. Die Auswertung der Testszenarien erfolgte abschließend sowohl grafisch als auch in Textform, wodurch ein leichtes Erkennen von Entwurfsfehlern und eine schnelle Analyse der Fehlerursache ermöglicht wurde.

In weiterführenden Arbeiten sollte zunächst die genaue Betrachtung der Übertragungsfehler durchgeführt werden. Durch experimentelle Tests kann eine Aussage über die Fehlerhäufigkeit getroffen werden. Unter der Verwendung dieser realitätsnahen Parameter kann das Gesamtsystem getestet und die korrekte Interaktion zwischen EMIF-Bridge und LVDS-Knoten sicher gestellt werden. In einem nächsten Schritt kann die Evaluierung der maximalen Knotenanzahl im Ring unter bestimmten Lastprofilen durchgeführt werden.

Neben der Beseitigung der in Abschnitt 8.3.1 beschriebenen Implementierungsfehler bietet sich eine Erweiterung des LVDS-Knotens an. Der LVDS-Knoten reagiert auf einen fatalen Fehler im LVDS-Ring mit dem Einstellen der Übertragung. Hier fehlt eine Rückmeldung an die EMIF-Bridge bzw. den DSP. Denkbar ist sowohl das Einführen eines neuen Interrupt-Ereignisses in der EMIF-Bridge als auch die Verwendung des *no-*

*de_change*-Ereignisses (siehe Tabelle 5.2).
Desweiteren fehlt die Möglichkeit, Sende- und Empfangspuffer des LVDS-Knotens zurückzusetzen. Die Resetsignale der EMIF-Bridge für die Sende- und Empfangslogik werden an Output-Ports ausgegeben und können entsprechend im LVDS-Knoten weiterverwendet werden.
Zur Beschleunigung der LVDS-Kommunikation bietet sich ein Bypass des Sendepuffers im LVDS-Knoten an. Da die Übertragung der Datenpakete von der EMIF-Bridge schnittstellengerecht durchgeführt wird, ist der Sendepuffer nur nötig, wenn die höher priorisierte LVDS-Paketweiterleitung ein direktes Senden in den Ring verhindert.

In einer weiterführenden Arbeit sollte das Gesamtsystem bestehend aus EMIF-Bridge und LVDS-Knoten in ein Quartus II-Projekt integriert und getestet werden. Wichtig für die Integration des Gesamtsystems ist die Timing Analyse in Quartus II. Diese ermöglicht Rückschlüsse auf ungünstige Logikpfade im Simulink Modell. Durch den Einbau von zusätzlichen Verzögerungselementen oder die Optimierung der Kombinatorik und funktionalen Blöcke kann das Zeitverhalten der Implementierung verbessert werden.
Zudem muss die Ursache für die fehlerhafte Synthese von Quartus II untersucht werden (siehe Abschnitt 8.2.5). Ein Ansatzpunkt ist hier die Analyse der Syntheseergebnisse mit unterschiedlichen Optimierungseinstellungen.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| CPLD | Complex Programmable Logic Device |
| DSP | Digitale Signalprozessor |
| EMIF | External Memory Interface |
| FPGA | Field Programmable Gate Array |
| FSM | Finite State Machine |
| HDL | Hardware Description Language |
| HPI | Hostport-Interface |
| ID | Identifier |
| IMR | Interrupt Mask Register |
| IRQ | Interrupt Request |
| IRR | Interrupt Request Register |
| ISR | Interrupt Service Routine |
| LED | Light-Emmitting Diode |
| LVDS | Low Voltage Differential Signaling |
| McBSP | Multichannel Buffered Serial Port |
| PIC | Programmable Interrupt Controller |
| RAM | Random Access Memory |
| RTL | Register Transfer Level |
| SFB | Sonderforschungsbereich |
| VHDL | Very High Speed Integrated Circuit Hardware Description Language |

# Literaturverzeichnis

[Alt08] ALTERA CORP.: ByteBlaster II Download Cable. User Guide, Juli 2008. – Online im Internet http://www.altera.com/literature/ug/ug_bbii.pdf, [Stand 02/2010]

[Alt10a] ALTERA CORP.: Cyclone II EP2C8 Q208. Product Website, März 2010. – Online im Internet http://www.altera.com/products/devices/cyclone2/overview/cy2-overview.html

[Alt10b] ALTERA CORP.: ModelSim-Altera Edition. Website, März 2010. – Online im Internet http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html

[Alt10c] ALTERA CORP.: Quartus II Web Edition. Website, März 2010. – Online im Internet http://www.altera.com/support/software/sof-quartus.html

[D.S04] D.SIGNT: NÖLKER, NORBERT ; KLEMENZ, ADOLF: D.Module.C6713 (Rev. 2.0). 1.1. Kerken : User Guide, September 2004. – Online im Internet: http://dsignt.de/save/dmodule/ugd6713.pdf, [Stand 01/2010]

[HH09] HALL, Stephen H. ; HECK, Howard L.: Advanced Signal Integrity for High-Speed Digital Designs. John Wiley & Sons, 2009

[Hir08] HIRSCH, Martin: Konzeption und Realisierung eines geschalteten seriellen Kommunikationssystems für ein Mehrprozessorsystem, Technische Universität Ilmenau, Diplomarbeit, Januar 2008

[Int88] INTEL CORP.: 8259A - Programmable Interrupt Controller. Data Sheet, Dezember 1988. – Online im Internet: http://pdos.csail.mit.edu/6.828/

2005/readings/hardware/8259A.pdf, [Stand 03/2010]

[Mül09] Müller, Michael: Hardware und Software für Kommunikationsaufgaben in einem DSP-Multiprozessorsystem mit programmierbarer Logik, Technische Universität Ilmenau, Diplomarbeit, Februar 2009

[Nat04] National Semiconductor Corp.: LVDS Owner's Manual. 3rd Edition. Manual, Januar 2004. – Online im Internet: http://www.national.com/appinfo/lvds/files/ownersmanual.pdf, [Stand 03/2010]

[Sch10] Schulze, Raik: Modellbasierte Entwicklung der Interprozessorkommunikation des Multi-DSP-Kompaktsystems, Technische Universität Ilmenau, Bachelorarbeit, Januar 2010

[Spe05] Spectrum Digital, Inc.: XDS510PP PLUS Parallel Port JTAG Emulator. User Guide, März 2005. – Online im Internet http://emulators.spectrumdigital.com/files/XDS510PPPLUS_UserGuide.pdf, [Stand 03/2010]

[Tex06] Texas Instruments Inc.: TMS320C6713B Floating-Point Digital Signal Processor (Doc-ID: SPRS294B). Dallas : Data Sheet, Juni 2006. – Online im Internet: http://focus.ti.com/lit/ds/symlink/tms320c6713b.pdf, [Stand 01/2010]

[Tex08] Texas Instruments Inc.: TMS320C6000 DSP - External Memory Interface (EMIF) (Doc-ID: SPRU266E). Dallas : Reference Guide, April 2008. – Online im Internet: http://focus.ti.com/lit/ug/spru266e/spru266e.pdf, [Stand 01/2010]

[Tex10] Texas Instruments Inc.: Code Composer Studio IDE. Website, März 2010. – Online im Internet http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html

[The10a] The MathWorks: HDL Coder. Website, März 2010. – Online im Internet http://www.mathworks.com/products/slhdlcoder/

[The10b] The MathWorks: Matlab. Website, März 2010. – Online im Internet http://www.mathworks.de/products/matlab/

[The10c] The MathWorks: Simulink. Website, März 2010. – Online im Internet http://www.mathworks.de/products/simulink/

[TU 10] TU Ilmenau: Sonderforschungsbereich 622 - Nanopositionier- und Nanomessmaschinen. Website, März 2010. – Online im Internet http://www4.tu-ilmenau.de/sfb622/index.htm

[Wik09] Wikimedia Foundation Inc.: Wikipediaartikel zum Hamming-Code. November 2009. – Online im Internet: http://de.wikipedia.org/wiki/Hamming-Code, [Stand 03/2010]

# Anhang

# A. Verwendete Hard- und Software

Hardware:

- DSP-Modul D.Module.C6713 [D.S04]
- Basisboard, Backplane (Stand Februar 2009)
- FPGA Cyclone II EP2C8 Q208 [Alt10a]
- ByteBlaster II [Alt08]
- DSP TMS320C6000 [Tex08]
- JTAG-Emulator XDS510PP Plus [Spe05]

Software:

- Matlab R2009b (Version 7.9) [The10b]
- Simulink R2009b (Version 7.4) [The10c]
- Simulink HDL Coder (Version 1.6) [The10a]
- ModelSim Altera Startet Edition 6.5b [Alt10b]
- Quartus II Web Edition 9.0 [Alt10c]
- CCStudio [Tex10]

# B. Die Begleit-CD

- /
  - CCStudio
    - APIs ..... Übernommene APIs von Michael Müller
      - FPGA-RAM
      - GPIO_Mini
    - CCStudio Workspace ..... Setup- und Workspace-Einstellungen für CCStudio
    - EMIF_Bridge_Test ..... DSP-Programm für den Integrationstest (siehe Abschnitt 8.2.4)
  - Diagramme ..... Selbst erstellte Blockschaltbilder, Aktivitätsdiagramme, Registerübersichten etc.
    - EMIF-Bridge
      - Implementierung
    - Modelltest
  - Dokumente ..... Datenblätter und User Guides
    - C6713
    - EMIF
    - HW-Setup
      - Eagle-Dateien
    - LVDS
    - Programmable Interrupt Controller (PIC)
    - Vorarbeiten
  - Latex ..... Latex- und PDF-Dateien dieser Arbeit
    - bib
    - figures
    - Latex_Template
    - tex

- /
  - Matlab
    - EMIF_Bridge — Matlab-Modell der EMIF-Bridge mit Testumgebung
      - hdlsrc — Generierte VHDL-Dateien
    - Komplettsystem — Simulation des Gesamtsystems (3 LVDS-Knoten mit jeweils einer EMIF-Bridge)
    - RAM Test
    - Simulation LVDS — Simulationsumgebung (siehe Abschnitt 5.2)
      - Ergebnisse — Matlab-Figures und Protokolle der Testdurchläufe (Abschnitt 8.3)
    - Simulationskomponenten — Matlab-Modelle für die Komponenten der Simulationsumgebung
      - Auswertung — M-File für die Auswertung der Protokolle
      - EMIF Signalgenerator — Generator für EMIF-Signale (Beschreibung im Modell)
      - Empfangsauswertung
      - Fehlergenerator
      - Paketgenerator
  - Modelsim
    - EMIF_Bridge_Test — ModelSim-Projekt und Testbench für EMIF-Bridge mit Testumgebung
    - RAM_Test
  - Quartus
    - EMIF_Bridge_Test — Quartus II Projekt für EMIF-Bridge mit Testumgebung
    - RAM_Test
  - Zwischenreviews — Dokumentation der Zwischenstände
  - LVDS_Package_Analyser.xlsm — Excel-Worksheet für die einfache Umformung des Paketheaders in eine dezimale Darstellung

# C. Ergänzende Darstellungen

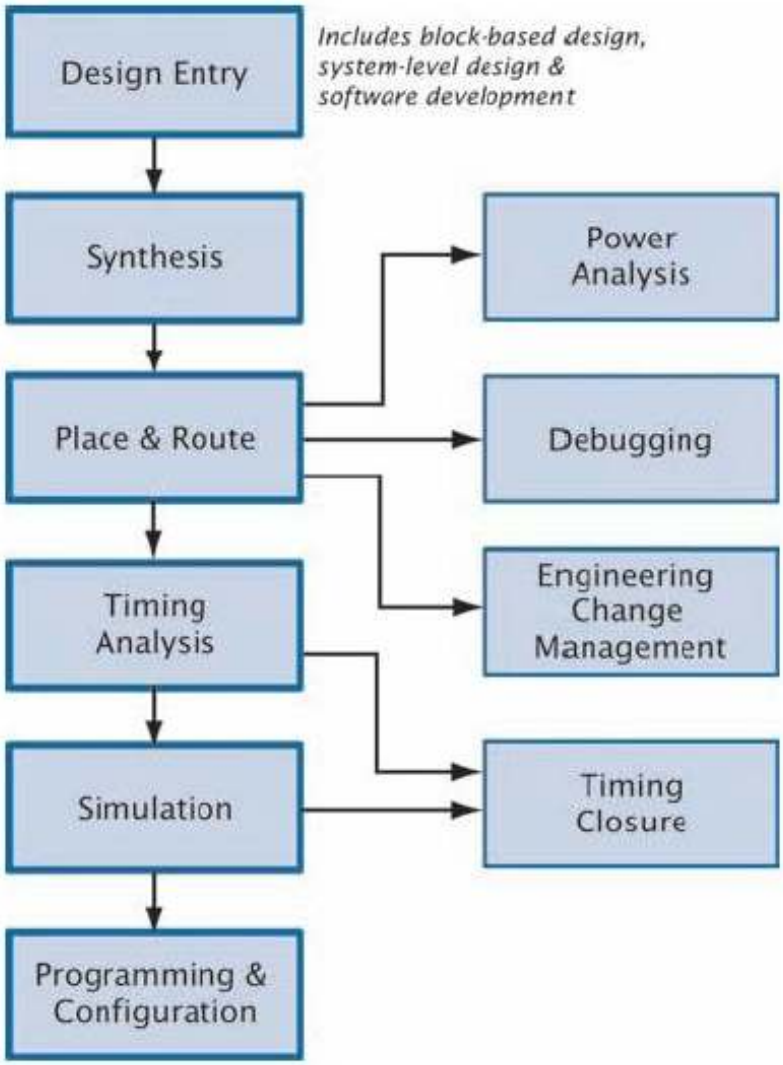



Abb. C.1.: Quartus Design Flow [Alt10c]

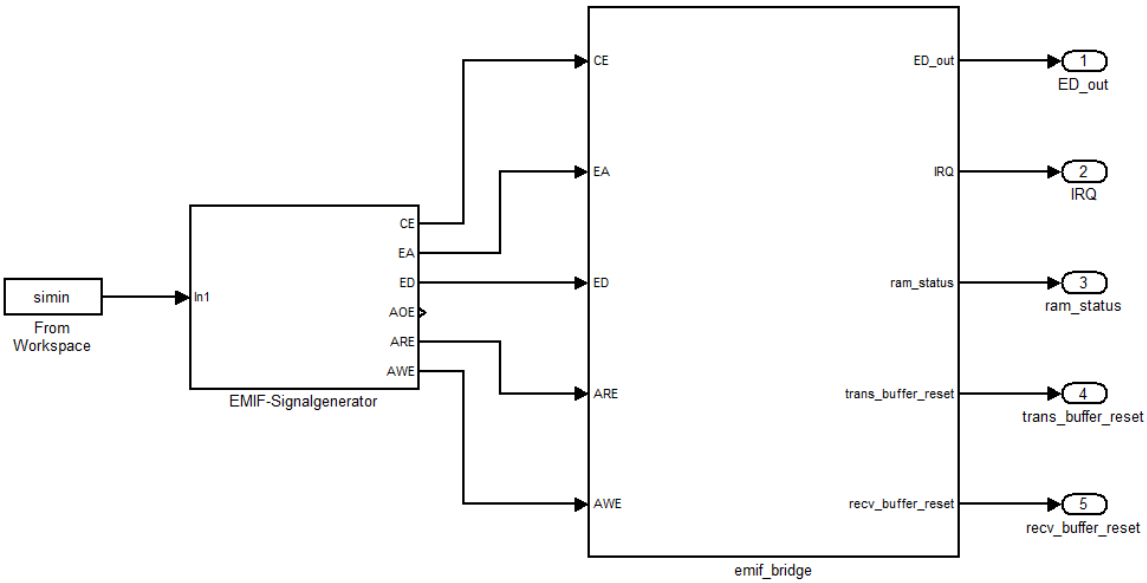



Abb. C.2.: EMIF-Bridge Test Bench

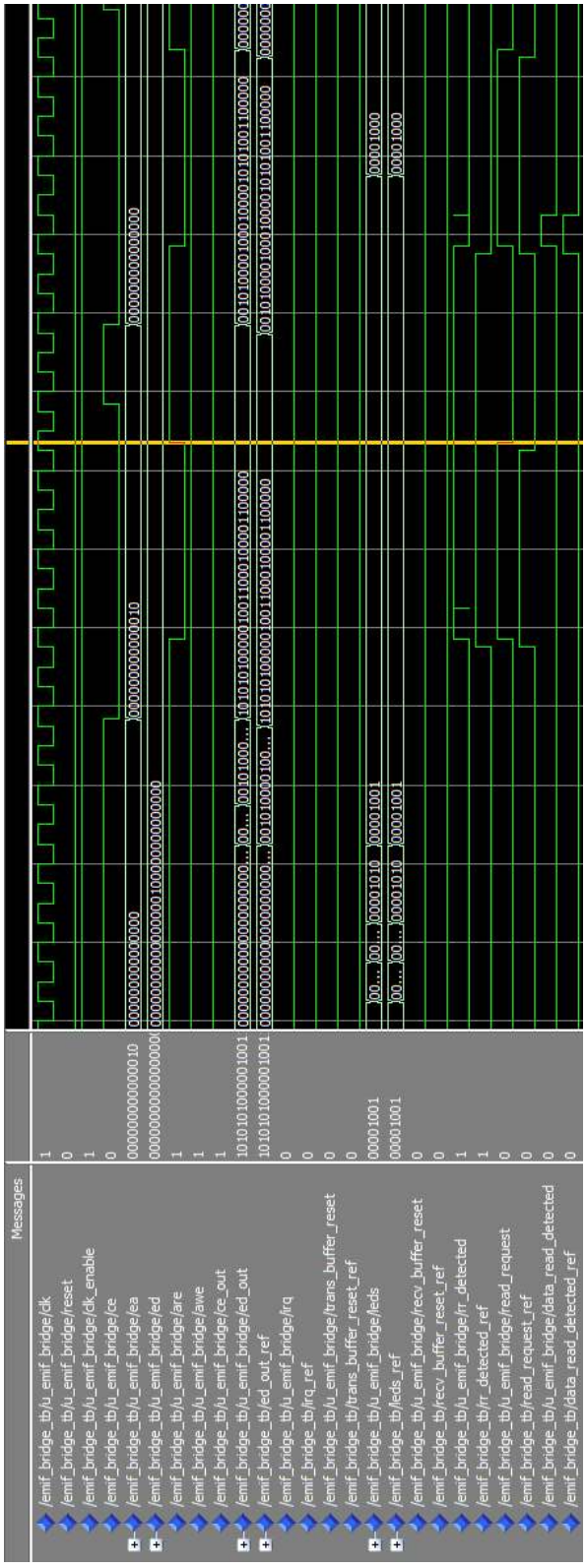



Abb. C.3.: ModelSim Simulationsergebnisse der EMIF-Bridge mit Testumgebung

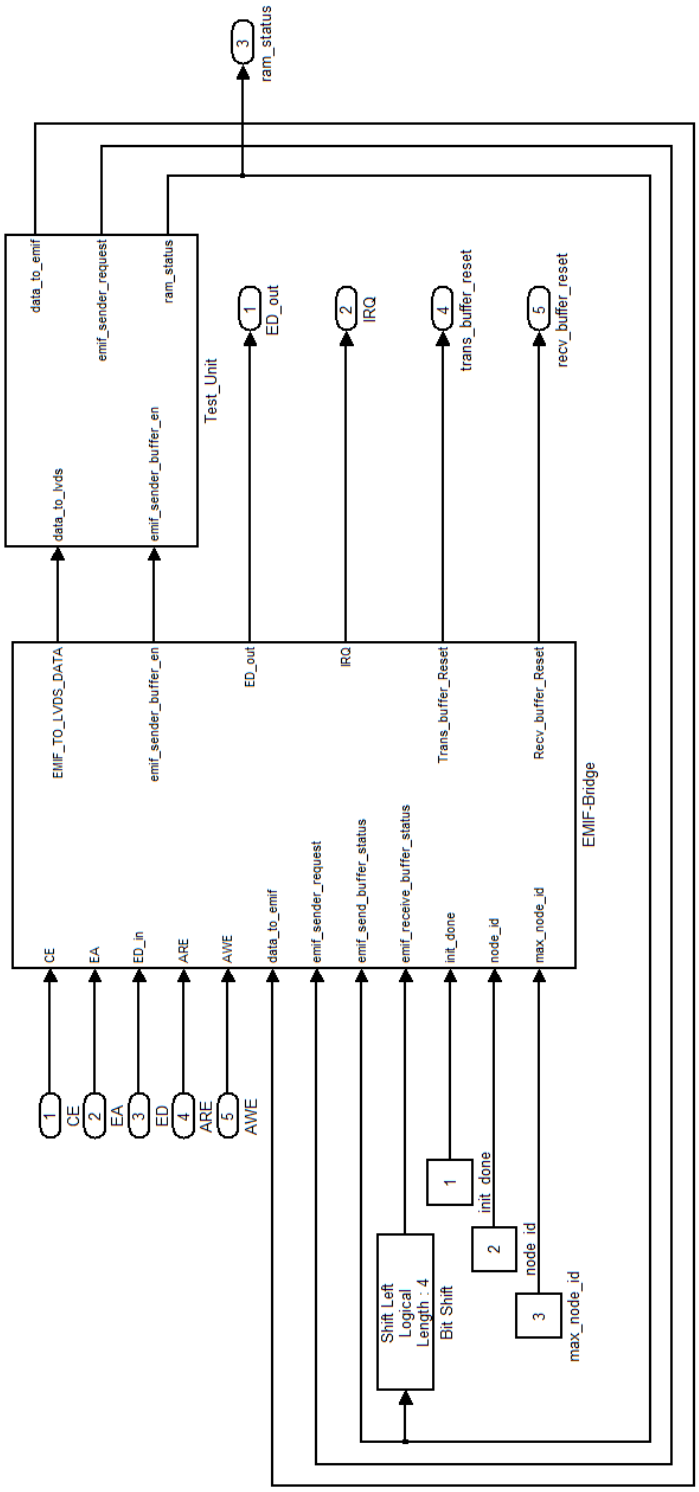



Abb. C.4.: EMIF-Bridge mit Testumgebung